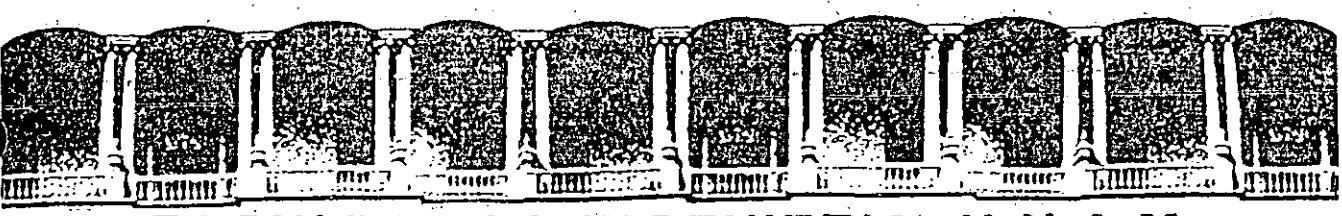
—DIRECTORIO DE PROFESORES DEL CURSO: USO AVANZADO DE MS-DOS—(PROGRAMACION BATCH)
DEL 11 AL 29 DE MAYO DE 1992.

ING. ANTONIO PEREZ AYALA (COORDINADOR)
GERENTE DE SISTEMAS
INGENIERIA EN PROYECTOS DE CONTROL
CALZ. DE LA VIGA No. 376 A Y B
COL. JAMAICA
15800 MEXICO, D.F.
TEL. 538-66-02

M. EN C. ALEJANDRO JIMENEZ GARCIA
GERENTE DE SISTEMAS DE INFORMACION
ORACLE MEXICO
EJERCITO NACIONAL No. 579, 1o. P.
MEXICO, D.F.
TEL. 255-31-66

ING. SALVADOR MEDINA MORAN
ASESOR EN COMPUTO
CENAPRED
AV. DELFIN MADRIGAL No. 665
PEDREGAL DE SANTO DOMINGO
04360 MEXICO, D.F.
TEL. 554-82-50

ING. FEDERICO MORALES FAVILA
JEFE DE ING. DE SISTEMAS
BANCO DE MEXICO
CONDESA No. 5, 4o. PISO
COL. CENTRO
MEXICO, D.F.
TEL. 518-05-00 EXT 2525



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

USO AVANZADO DE MS-DOS

PROGRAMACION BATCH

MATERIAL DIDACTICO

MAYO, 1992.

USO AVANZADO DEL SISTEMA OPERATIVO MS-DOS
(PROGRAMACION BATCH)

C O N T E N I D O

INTRODUCCION

Este documento tiene el objetivo de describir las capacidades del proceso de archivos BATCH proporcionadas originalmente por el sistema operativo MS-DOS. Debido a ello, aquí se describirán ciertos aspectos del lenguaje BATCH en forma mas detallada y extensa que en muchos otros textos sobre este tema.

Sin embargo, este manual no pretende ser uno mas sobre el uso del sistema operativo MS-DOS, pues los libros sobre este tema ya abundan; por el contrario, estas notas tratan específicamente de la programación BATCH por lo que deben centrar su atención en este punto. Debido a ello en este texto no se incluirá ninguna descripción detallada sobre conceptos básicos de uso del sistema operativo, como serían nombres de archivos, los comandos básicos de manejo de archivos y directorios, nombres de dispositivos, redireccionamiento, etc. Por lo tanto, en este documento se supone que el lector ya cuenta con conocimientos básicos generales de uso y operación del sistema operativo MS-DOS.

En forma similar, en las descripciones de los comandos de MS-DOS aquí presentadas invariablemente se tomará el punto de vista del proceso BATCH. Esto quiere decir que si un cierto comando se puede aprovechar en forma diferente al incluirse dentro de un programa BATCH que al ejecutarse directamente desde el teclado, sólo la primera de estas dos posibilidades será descrita, a menos que el comando en cuestión esté directamente relacionado con el proceso BATCH en cuyo caso se describirá por completo.

Si el lector desea ampliar sus conocimientos sobre algún comando en particular o sobre detalles específicos de MS-DOS no incluídos en este documento se le recomienda consultar alguno de los libros al respecto, de los cuales se incluye una amplia bibliografía al final de estas notas, o bien, directamente en el manual del sistema operativo MS-DOS de su computadora el cual, por supuesto, es la fuente original de información de estas características.

Capítulo 1 - COMANDOS BATCH BASICOS

1.1 Archivos de comandos_BATCH

El sistema operativo MS-DOS incluye una serie de características orientadas a facilitar y controlar la ejecución de procesos rutinarios y repetitivos. Todas estas facilidades están basadas en lo que el manual de MS-DOS denomina en inglés "BATCH files", esto es, "archivos BATCH", o bien, "archivos de comandos BATCH".

La idea básica de los archivos BATCH es el poder activar en forma sencilla una serie de procesos que frecuentemente se ejecutan de la misma manera, como sería formatear un disco nuevo, ejecutar un cierto paquete o sistema, hacer una protección de archivos, etc.

En lugar de teclear uno por uno los comandos correspondientes para realizar estos procesos cada vez que se necesitan éstos se pueden agrupar dentro de un archivo BATCH, el cual se definirá una sola vez pero podrá ejecutarse muchas. El archivo debe tener la extensión BAT y podrá crearse con cualquier editor de texto, por ejemplo, con el editor EDLIN incluído con MS-DOS.

Para ejecutar al archivo BATCH se debe teclear su nombre sin incluir la extensión; la ejecución del archivo de comandos BATCH implica la ejecución de los comandos de MS-DOS contenidos en él. Por ejemplo, suponga que para obtener un nuevo disco del sistema operativo se ejecutan los siguientes cuatro comandos:

```
A> FORMAT B: /S

A> COPY CONFIG.SYS B:

A> COPY AUTOEXEC.BAT B:

A> CHKDSK B:
```

Es decir, cada vez que se necesita un nuevo disco del sistema operativo se inserta éste en la unidad B: y se teclean los cuatro comandos anteriores, ejecutándolos uno tras de otro.

Estos cuatro comandos se pueden agrupar dentro de un archivo de comandos BATCH llamado, por ejemplo, NEWDISK.BAT; es decir, el archivo NEWDISK.BAT se podría generar con cualquier editor de texto o con el comando COPY CON: NEWDISK.BAT de MS-DOS, y su contenido serían las siguientes cuatro líneas:

```
FORMAT B: /S
COPY CONFIG.SYS B:
COPY AUTOEXEC.BAT B:
CHKDSK B:
```

Una vez agrupados estos cuatro comandos en este archivo, se pueden ejecutar todas las veces que sea necesario simplemente tecleando el nombre del archivo BATCH que los contiene. En este caso bastaría con teclar NEWDISK de la siguiente forma:

```
A> NEWDISK
```

Un archivo de comandos BATCH puede contener cualquier comando de MS-DOS que sea válido ejecutar directamente desde el teclado, es decir, todos los comandos internos y externos descritos en el manual de MS-DOS, así como cualquier paquete adicional que se encuentre disponible en la computadora, como sería un editor de texto, una base de datos, una hoja de cálculo electrónico, etc.

Además de los comandos de uso común, el sistema operativo MS-DOS proporciona una serie de facilidades y comandos adicionales diseñados para aumentar la capacidad de los archivos de comandos BATCH; ejemplos de estos comandos son: PAUSE, GOTO, IF, SHIFT, etc. Puesto que todos estos comandos se colocan dentro de un archivo BATCH para su posterior ejecución, podemos decir que ellos conforman las instrucciones del "lenguaje de programación BATCH", y desde este punto de vista, a un "archivo BATCH" también se le puede llamar "programa BATCH". En este manual se utilizarán indistintamente cualquiera de los dos términos anteriores.

1.2 Creación de programas

De acuerdo con la descripción anterior, un programa BATCH está formado por una serie de comandos de MS-DOS, los cuales primero se incluyen dentro de un archivo y posteriormente se ejecutan. Desde este punto de vista, la creación de un archivo de comandos BATCH es por completo similar a la creación de un programa en cualquier otro lenguaje de programación, BASIC por ejemplo.

Por lo tanto, el proceso mediante el cual se crea un archivo de comandos BATCH se puede llamar "programación". Existen numerosos métodos de enseñanza que pretenden que el lector aprenda a "programar" utilizando como base algún lenguaje de programación popular, usualmente BASIC. Este manual sería muy ambicioso y poco realista si pretendiera enseñarle programación al lector sin experiencia en computación; sin embargo, es conveniente mencionar el siguiente par de puntos al respecto.

La creación de programas pequeños y simples puede parecer muy sencilla y sin importancia, sin embargo, es precisamente mediante el desarrollo de múltiples programas de un grado creciente de dificultad como se obtiene experiencia en programación y no mediante la lectura de una serie de libros. Por lo tanto, se recomienda al lector que a medida que se introduzca en nuevos temas vaya desarrollando varios programas BATCH similares a los ejemplos mostrados, por muy sencillos que éstos pudieran parecer.

El desarrollo de un programa grande y complicado es una labor que forzosamente requiere amplia experiencia; sin embargo, además de la experiencia existen diversos métodos que facilitarán la creación de un programa de este tipo. La parte central de esos métodos se puede resumir en las tres etapas que se describen a continuación, por lo que será recomendable seguirlas durante el desarrollo de cualquier programa relativamente grande.

1.- Análisis del problema.

En esta etapa se debe analizar todo aspecto relacionado con el programa por desarrollar buscando sobre todo responder las dos siguientes preguntas: ¿Que se desea hacer?, esto es, establecer perfectamente los objetivos del nuevo programa, y ¿Como se hará?, es decir, definir sin ninguna duda posible los pasos generales que se deben realizar para conseguir los objetivos establecidos.

El análisis es la base del nuevo programa, ya que si éste revela que se requiere un cierto resultado que no se puede obtener en la computadora, implicará que no tiene caso desarrollar el programa de esa forma, puesto que nunca se obtendrá de él lo que se esperaba. Jamás se debe pasar a la siguiente etapa si no se ha concluído por completo el análisis, ya que hasta un pequeño detalle no resuelto en esta etapa puede ocasionar complicaciones muy grandes en las etapas posteriores.

2.- Diseño del programa.

En esta etapa se debe desglosar el resultado del análisis en términos del programa de computadora, es decir, para cada uno de los pasos generales definidos por el análisis se debe especificar el comando o comandos encargados de efectuar esos pasos.

Estas especificaciones deben incluir todos los detalles relacionados con el programa en sí, como serían: cuáles son los comandos que se utilizarán, la forma de ejecutarlos, el orden correcto en que deben ejecutarse, los nombres de las variables auxiliares y el valor que contendrá cada una de ellas, si se dividirá una sección grande del programa en secciones más pequeñas y el nombre que llevará cada una de ellas, etcétera.

Por supuesto, la mayoría de estos detalles dependen de la experiencia y conocimiento que se tenga del lenguaje en sí, por lo que mientras mayor sea la experiencia con el lenguaje de programación se harán diseños mejores y sin errores.

3.- Programación.

Si las dos etapas anteriores se realizaron correctamente, esta etapa debe ser la mas sencilla de todas y consiste simplemente en traducir las especificaciones del diseño a las instrucciones correspondientes que formarán el nuevo programa, es decir, esta etapa consiste en la escritura en sí del nuevo programa.

Hay que tener presente que en donde efectivamente se resolvió el problema fué en la etapa del análisis y se terminó de pulir en la de diseño, mientras que la etapa de programación es puramente mecánica. Por supuesto, si se omiten las dos primeras etapas y se pretende desarrollar un programa relativamente grande escribiéndolo directamente en la computadora, las complicaciones de hacer las tres etapas al mismo tiempo, sin planeación y sobre la marcha, ocasionarán grandes problemas en el desarrollo que frecuentemente ocasionarán errores en su funcionamiento.

En conclusión, conviene comenzar por lo sencillo, hacer muchos ejercicios de programación e ir avanzando hacia desarrollos y temas cada vez mas complicados. En cuanto nos topemos con el primer ejercicio lo suficientemente difícil como para no poder hacerlo de esta forma, habrá que desarrollarlo mediante las tres etapas antes vistas y a partir de ahí en adelante utilizarlas siempre, lo cual nos dará cada vez mas experiencia en su uso.

El lector que haga esto, sin duda alguna aprenderá programación.

1.3 Comando REM

Forma de uso: REM comentario

Permite insertar comentarios en el programa.

Si un programa BATCH es muy grande, o utiliza algunos comandos poco comunes, puede ser difícil comprender qué es lo que pretende efectuar un cierto comando si se está revisando el programa BATCH tiempo después de haberlo desarrollado. Por esta razón, es conveniente incluir comentarios que describan algunas partes del programa particularmente difíciles o confusas. Los comentarios no tienen ningún efecto durante la ejecución de un programa BATCH.

Programa de ejemplo: EJ-REM.BAT

```
REM SE FORMATEA EL DISCO INCLUYENDO LA OPCION "/System"
FORMAT B: /S
REM SE COPIAN LOS ARCHIVOS QUE UTILIZA EL SISTEMA MS-DOS
COPY CONFIG.SYS B:
COPY AUTOEXEC.BAT B:
REM SE REVISA EL NUEVO DISCO UNA VEZ TERMINADA LA COPIA
CHKDSK B:
```

1.4 Comando ECHO

```
                  ON
Forma de uso: ECHO OFF
                  mensaje
```

Activa o desactiva el eco, o muestra un mensaje en la pantalla.

Cuando se ejecuta un programa BATCH, el sistema operativo MS-DOS muestra en la pantalla cada uno de los comandos contenidos en el archivo a medida que los va ejecutando. Esta característica, llamada "eco" ("echo" en inglés), es conveniente si el programa BATCH se está desarrollando y probando, pero si un programa ya está terminado, el continuo mostrado de cada uno de sus comandos no es agradable. Para desactivar el mostrado de los comandos utilice ECHO OFF; para volver a activarlo utilice ECHO ON.

Cuando el ECHO está en OFF en la pantalla no aparecerá ninguno de los comandos que se ejecuten, por lo que para mostrar algún mensaje en la pantalla desde un programa BATCH en ejecución se deberá utilizar también el comando ECHO, pero incluyendo el mensaje que se desea mostrar en lugar de las palabras ON u OFF.

Programa de ejemplo: EJ-ECHO1.BAT

```
REM ESTE COMANDO APARECE EN LA PANTALLA ANTES DE EJECUTARSE
ECHO EN ESTE CASO, APARECE EL COMANDO Y EL MENSAJE MOSTRADO
REM EL SIGUIENTE COMANDO APAGA EL "ECO"
ECHO OFF
REM ESTE COMANDO YA NO APARECE EN LA PANTALLA
ECHO ESTE MENSAJE APARECE SIN MOSTRAR EL COMANDO QUE LO GENERO
REM NUEVAMENTE SE ACTIVA EL "ECO"
ECHO ON
REM NUEVAMENTE APARECE ESTE COMANDO EN LA PANTALLA
```

Si el comando ECHO se ejecuta sin incluir ningún caracter enseguida de él o incluyendo solamente espacios en blanco, el comando mostrará en la pantalla el estado actual del eco, es decir, las palabras ON u OFF, por lo que si se desea mostrar una línea en blanco en la pantalla se debe incluir un caracter especial en el comando ECHO. Este caracter se puede obtener desde MS-DOS, o bien, desde el editor EDLIN oprimiendo la tecla de función F7, lo cual mostrará en la pantalla el par de caracteres "^@" como una indicación de que se oprimió la tecla F7.

Hay que tener presente que si se teclea el caracter "^" seguido de "@" no se obtendrá el mismo efecto aunque en la pantalla aparezca en forma igual, por lo que es necesario precisamente oprimir la tecla F7. Si se desea obtener este efecto desde otro editor de texto que no sea EDLIN, entonces deberá hacerse lo siguiente: manteniendo oprimida la tecla ALT se deberá teclear el número 255 en el teclado numérico y al terminar soltar la tecla ALT; esto aparentemente generará un espacio en blanco que, sin embargo, ocasionará que el comando ECHO muestre una línea en blanco en lugar de las palabras ON u OFF.

Programa de ejemplo: EJ-ECH02.BAT

```
ECHO OFF
ECHO ESTA LINEA APARECE EN LA PANTALLA
REM EL SIGUIENTE COMANDO DEJA UNA LINEA EN BLANCO
REM RECUERDE QUE EL COMANDO ECHO DEBE TERMINAR CON LA TECLA F7
REM Y NO CON EL PAR DE CARACTERES "^" SEGUIDO DE "@"
ECHO ^@
ECHO LINEA SEPARADA DE LA ANTERIOR POR UNA LINEA EN BLANCO
```

1.5 Comando CLS

Forma de uso: CLS

Borra la pantalla.

En ciertos programas BATCH es conveniente borrar la pantalla para comenzar a mostrar la información en una pantalla limpia; esto no es necesario si el programa BATCH es pequeño o muestra poca información. Si va a utilizar CLS, ponga un comando ECHO OFF al principio del programa y coloque a CLS después de él.

Programa de ejemplo: EJ-CLS.BAT

```
ECHO OFF
CLS
REM TODOS LOS COMANDOS SIGUIENTES TERMINAN CON LA TECLA F7
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO          ESTE MENSAJE APARECE EN LA MITAD DE LA PANTALLA
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
ECHO ^@
```

1.6 Comando PAUSE

Forma de uso: PAUSE

Detiene la ejecución del programa.

El comando PAUSE detiene la ejecución de un programa BATCH permitiendo que el usuario realice alguna labor, por ejemplo un cambio de disco, y continúe después con el programa al oprimir cualquier tecla. Si se desea cancelar la ejecución del programa BATCH, se debe oprimir la tecla Ctrl-C la cual también cancelará la ejecución de un programa BATCH si se oprime en cualquier momento de su ejecución. (Ctrl-C indica mantener oprimida la tecla marcada Ctrl, y oprimir entonces la tecla de la letra C)

Siempre que se utilice el comando PAUSE deberá incluirse un comando ECHO inmediatamente antes que muestre un mensaje que describa el motivo de la pausa, con el fin de que el usuario sepa qué debe hacer para que el programa continúe correctamente.

Programa de ejemplo: EJ-PAUSE.BAT

```
ECHO OFF
CLS
REM SE SOLICITA INSERTAR EL NUEVO DISCO
ECHO INSERTE EL NUEVO DISCO EN LA UNIDAD B:
PAUSE
REM SE FORMATEA EL DISCO INCLUYENDO LA OPCION "/System"
FORMAT B: /S
REM SE COPIAN LOS ARCHIVOS QUE UTILIZA EL SISTEMA
COPY CONFIG.SYS B:
COPY AUTOEXEC.BAT B:
REM SE REVISA EL NUEVO DISCO UNA VEZ COPIADOS LOS ARCHIVOS
CHKDSK B:
```

Capitulo 2 - FACILIDADES BATCH AVANZADAS

2.1 Uso del caracter tanto por ciento

Varias de las facilidades BATCH descritas en este capítulo y en algunos capítulos posteriores utilizan el caracter tanto por ciento (%) como una clave para indicar el uso de éstas, es decir, existe un cierto caracter cuyo uso está reservado por MS-DOS con el fin de tener acceso a una serie de facilidades BATCH avanzadas; tal caracter es el tanto por ciento.

Debido a esto el caracter tanto por ciento no puede ser utilizado libremente por el usuario para sus propios fines, por lo que en caso de que el usuario forzosamente necesite incluir un tanto por ciento en un programa BATCH, por ejemplo en un nombre de archivo, el tanto por ciento deberá colocarse dos veces seguidas.

Por ejemplo, si existiera un archivo llamado IVA15%.DAT, para incluir su nombre dentro de un archivo de comandos BATCH éste se debe teclear de la siguiente forma: IVA15%%.DAT. En todo caso, lo mas recomendable es evitar el uso del tanto por ciento en cualquier nombre de archivo definido por el usuario.

Es muy importante hacer notar que esta característica se aplica exclusivamente a los comandos que se hayan colocado dentro de un archivo de comandos BATCH y de ninguna manera a los comandos que pudieran ejecutarse directamente por el teclado. Debido a ello, todas las descripciones de los comandos incluídas en este manual presuponen que el comando en cuestión está colocado dentro de un programa BATCH, por lo que si se desea utilizar un cierto comando directamente desde el teclado convendrá recordar lo siguiente.

Si en la descripción de un determinado comando se muestra un par de caracteres tanto por ciento inmediatamente juntos (%%), tal facilidad podrá ser utilizada directamente desde el teclado sustituyendo los dos tantos por ciento por uno sólo. Si en la descripción se muestra un sólo caracter tanto por ciento (%), tal facilidad no podrá utilizarse directamente desde el teclado.

2.2 Parámetros del programa BATCH

Los programas BATCH permiten ejecutar en forma automática una serie de procesos previamente definidos; sin embargo, la potencia real de esta facilidad estriba en el hecho de que estos procesos se puedan ejecutar sobre diferentes datos en distintas ejecuciones. Esto puede hacerse mediante la siguiente facilidad.

Todos los comandos de MS-DOS se activan de una misma manera, llamada "formato general", que consiste en lo siguiente:

```
COMANDO PARAMETRO1 PARAMETRO2 ...
```

La palabra COMANDO se refiere al nombre del comando en sí, mientras que PARAMETRO1, PARAMETRO2, etc. son los "parámetros" del comando, esto es, son palabras adicionales que le indican al comando los datos sobre los que va a operar.

Por ejemplo, si consideramos el siguiente comando COPY:

```
COPY ARCHIVO.DAT B:PROTECC.DAT /V
```

tendríamos que el comando es COPY, su primer parámetro es ARCHIVO.DAT, el segundo es B:PROTECC.DAT y el tercero es /V.

En forma similar, cuando se activa un archivo de comandos BATCH se pueden proporcionar una serie de parámetros, los cuales podrán ser tomados dentro del programa BATCH utilizando el caracter tanto por ciento (%) en la forma descrita a continuación.

Si dentro del archivo de comandos BATCH aparece el tanto por ciento seguido de inmediato por un dígito (p.e. %1 %2 etc.), los dos caracteres serán sustituídos por el parámetro colocado en la posición indicada por el dígito, esto es, %1 se cambiará por el primer parámetro proporcionado, %2 por el segundo, etc. Esto permite tomar hasta nueve parámetros, desde el %1 hasta el %9.

Adicionalmente, el par de caracteres %0 serán sustituídos por el nombre del archivo de comandos BATCH mismo que se esté ejecutando. Si se utiliza el número de un parámetro que no se proporcionó (p.e. %6 cuando sólo se dieron 4) ambos caracteres serán eliminados, no quedando nada en su lugar.

Por ejemplo, si se tuviera un archivo de comandos BATCH llamado EJEMPLO.BAT, el cual hubiera sido activado desde MS-DOS mediante la línea mostrada a continuación, dentro del programa EJEMPLO.BAT el par de caracteres tantoporciento-dígito se sustituirían por las palabras mostradas en seguida:

```
A> EJEMPLO REVISA DATOS.DAT /T

%0 = EJEMPLO
%1 = REVISA
%2 = DATOS.DAT
%3 = /T
%4 hasta %9 =
```

A continuación se muestra un pequeño programa de ejemplo de uso de parámetros.

Programa de ejemplo: EJ-PARS.BAT

```
ECHO OFF
CLS
ECHO YO SOY EL PROGRAMA %0
ECHO ESTOS SON MIS PARAMETROS
ECHO EL PRIMERO: "%1"
ECHO EL SEGUNDO: "%2"
ECHO EL TERCERO: "%3"
ECHO EL CUARTO: "%4"
ECHO Y EL QUINTO: "%5"
ECHO FIN DEL EJEMPLO
```

Si el programa anterior se ejecutara mediante lo siguiente:

```
A> EJ-PARS EJEMPLO DE PARAMETROS
```

mostraría en la pantalla las siguientes líneas:

```
YO SOY EL PROGRAMA EJ-PARS
ESTOS SON MIS PARAMETROS
EL PRIMERO: "EJEMPLO"
EL SEGUNDO: "DE"
EL TERCERO: "PARAMETROS"
EL CUARTO: ""
Y EL QUINTO: ""
FIN DEL EJEMPLO
```

2.3 Comando GOTO y etiquetas

Forma de uso: GOTO etiqueta

Pasa a otra línea del programa.

Este comando, junto con el comando IF descrito en seguida, son las piezas fundamentales del lenguaje de programación BATCH, ya que el comando GOTO permite alterar la ejecución normalmente secuencial de los comandos contenidos en un programa BATCH con el fin de transferir el orden de ejecución hacia algún comando colocado en otro lugar, lo cual permite la utilización repetida de un mismo comando varias veces, la ejecución o no ejecución de grupos de comandos seleccionados mediante el comando IF, etc.

La etiqueta colocada después del comando GOTO debe ser una palabra de un máximo de 8 letras, la cual deberá estar colocada en otra línea del programa BATCH que comience con el caracter dos puntos (:) seguido inmediatamente por la etiqueta, esto es, no debe haber ningún espacio en blanco al principio de la línea, ni entre el caracter (:) y la etiqueta.

Programa de ejemplo: EJ-GOTO1.BAT

```
REM LA EJECUCION COMIENZA CON EL PRIMER COMANDO Y
REM CONTINUA SECUENCIALMENTE CON LOS SIGUIENTES.
REM EL COMANDO "GOTO" ALTERA ESTE ORDEN DE EJECUCION:
GOTO OTRALINE
REM ESTOS COMANDOS NUNCA SERAN EJECUTADOS, DEBIDO A
REM QUE LA EJECUCION SE TRANSFIRIO A LA LINEA "OTRALINE"
:OTRALINE
REM LA EJECUCION CONTINUA EN ESTE PUNTO.
```

Para utilizar adecuadamente al comando GOTO éste debe combinarse con el comando IF, pues de lo contrario sólo se obtienen secciones de programa que nunca se ejecutan, como es el caso del ejemplo anterior, o secciones de programa que se repiten una y otra vez sin terminar nunca, como en el siguiente ejemplo.

Programa de ejemplo: EJ-GOTO2.BAT

```
ECHO OFF
CLS
REM ESTOS COMANDOS SE EJECUTAN UNA SOLA VEZ
REM AL INICIAR EL PROGRAMA
:OTRAVEZ
REM ESTOS COMANDOS SE EJECUTARAN UNA Y OTRA VEZ
REM EN FORMA REPETITIVA, SIN TERMINAR NUNCA
ECHO PARA CANCELAR LA EJECUCION DE ESTE PROGRAMA
ECHO OPRIMA LA TECLA CTRL-C
GOTO OTRAVEZ
```

2.4 Comando IF

```
                        palabra1 == palabra2
Forma de uso: IF NOT EXIST archivo.ext    comando
                        ERRORLEVEL #nivel
```

Ejecuta un comando condicionalmente.

El comando IF es uno de los comandos BATCH mas importantes, ya que nos permite la "ejecución condicional" de otro comando, es decir, la posibilidad de ejecutar o no ejecutar un cierto comando dependiendo de una condición establecida. Sin esta capacidad los programas BATCH se limitarían a ejecutar siempre los mismos comandos en el mismo orden; el comando IF, al combinarse con el comando GOTO antes visto, permite ejecutar en ciertas ocasiones unas partes de un programa y en ocasiones otras, enriqueciendo las capacidades y alcances de los programas BATCH.

El comando IF tiene tres formas diferentes de uso de acuerdo a tres condiciones distintas que puede establecer, que son:

1) Preguntar si dos palabras son iguales. En este caso se coloca el comando IF seguido por la primer palabra, seguida por el signo de igual dos veces (==), seguido por la segunda palabra, seguido por el comando de MS-DOS que se desea ejecutar cuando ambas palabras sean iguales. Por ejemplo, para mostrar el archivo indicado por el segundo parámetro en el caso de que el primero haya sido la palabra MUESTRA, se usaría lo siguiente:

```
IF %1 == MUESTRA TYPE %2
```

   Desgraciadamente, el comando IF sólo considerará iguales dos palabras si están formadas exactamente por las mismas letras, considerando diferentes una letra mayúscula de una minúscula. Puesto que los comandos de MS-DOS se pueden dar en letras mayúsculas o minúsculas en forma indistinta, se acostumbra repetir el comando IF dos veces, una con la palabra en mayúsculas y la otra en minúsculas. Por ejemplo:

```
IF %1 == MUESTRA TYPE %2
IF %1 == muestra TYPE %2
```

   Sin embargo, si cualquier letra de una palabra tiene diferente tamaño que las otras (p.e. Muestra, MUEsTRA, etc.), ninguno de los comandos IF anteriores la identificarán.

2) Preguntar si un cierto archivo existe en el disco. En este caso se coloca el comando IF seguido por la palabra EXIST, seguido por el nombre del archivo, seguido por el comando por ejecutar en el caso de que el archivo efectivamente exista.

```
IF EXIST %1.BAK DEL %1.BAK
```

El comando anterior borraría el archivo cuyo nombre lo dá el primer parámetro y con extensión .BAK en el caso de que éste existiera; en caso contrario, no se intentará borrarlo.

3) Preguntar por el valor entregado por el último proceso ejecutado. En MS-DOS cuando los procesos externos terminan su ejecución proporcionan un cierto valor, denominado ERRORLEVEL, el cual se puede utilizar para saber si tal proceso terminó debido a un error o terminó correctamente. El estándar establecido por MS-DOS dice que si el ERRORLEVEL entregado vale cero, el proceso terminó correctamente, mientras que si es un valor diferente de cero terminó debido a algún error.

Para preguntar por este valor se coloca el comando IF seguido de la palabra ERROLEVEL, seguido de un número, seguido del comando por ejecutar en caso de que el ERRORLEVEL entregado sea mayor o igual al número dado. Por ejemplo, suponga que un programa, llamado PROCESO, entrega un ERRORLEVEL igual a 2 si el programa se canceló debido a un error fatal, un valor de 1 si se presentó alguna situación no fatal que requiera de una revisión posterior de los resultados obtenidos, y un valor de cero si terminó correctamente. El siguiente programa BATCH ejecutaría a PROCESO y mostraría el mensaje adecuado:

```
ECHO OFF
CLS
PROCESO
IF ERRORLEVEL 2 GOTO FATAL
IF ERRORLEVEL 1 GOTO REVISA
ECHO TERMINO CORRECTAMENTE
GOTO FIN
:REVISA
ECHO PRECAUCION: REVISAR LOS RESULTADOS
GOTO FIN
:FATAL
ECHO ERROR FATAL, PROCESO CANCELADO
:FIN
```

Hay que tener presente que esta forma del comando IF pregunta si el ERRORLEVEL es mayor o igual que el número dado, razón por la cual estamos obligados a hacer las preguntas en orden descendente respecto al ERRORLEVEL cuando el programa ejecutado pueda proporcionar varios valores diferentes. Por ejemplo, si en el programa anterior se hubiera preguntado primero por el valor de 1, se hubiera ejecutado el GOTO correspondiente cuando el ERRORLEVEL hubiera sido 1 o 2, es decir, cuando hubiera sido un valor mayor o igual a 1.

El ERRORLEVEL sólo lo puede modificar un comando externo, esto es, un comando que exista en disco en forma de archivo con la extensión .COM o .EXE, por lo que la ejecución de cualquier comando interno no afectará al ERRORLEVEL que haya sido fijado por el último comando externo ejecutado. El ERRORLEVEL puede tener un valor mínimo de cero y un valor máximo de 255.

El comando IF tiene adicionalmente la posibilidad de incluir la palabra NOT antes de la condición que establece, lo cual permite preguntar por la condición contraria a las tres antes descritas. Debido a esto, se cuenta además con las siguientes condiciones:

4) Preguntar si dos palabras son diferentes, por ejemplo:

```
IF NOT %2 == BORRA GOTO NO-BORRA
```

El comando anterior transferirá la ejecución a la etiqueta NO-BORRA si el segundo parámetro no es la palabra BORRA.

5) Preguntar si un cierto archivo no existe en el disco. P.e.

```
IF NOT EXIST B:%1 COPY %1 B:
```

El comando anterior copiará el archivo indicado por el primer parámetro al disco colocado en la unidad B: cuando tal archivo no exista en ese disco.

6) Preguntar si el valor del ERRORLEVEL entregado por el último proceso es menor que un número dado. Por ejemplo:

```
IF NOT ERRORLEVEL 4 GOTO NO-GRAVE
```

El comando anterior transferirá la ejecución a la etiqueta NO-GRAVE cuando el ERRORLEVEL proporcionado por el último comando externo ejecutado sea 0, 1, 2 o 3, esto es, cualquier valor menor a 4.

Si bien no se incluye ningún programa de ejemplo sobre el comando IF, este comando es imprescindible para crear programas BATCH de cierta complejidad, por lo que habrá suficientes ejemplos de uso del comando IF en casi todos los programas de ejemplo restantes.

2.5 Comando SHIFT

Forma de uso: SHIFT

Desplaza la posición de los parámetros del programa.

Normalmente los parámetros de un archivo de comandos BATCH se toman mediante el caracter tanto por ciento seguido por un dígito que representa al parámetro, por lo que %1 indica al primer parámetro, %2 al segundo, y así hasta el noveno indicado por %9.

Cada ejecución del comando SHIFT desplaza esta posición relativa de tal forma que después de la primera ejecución de SHIFT, %1 representa al segundo parámetro, %2 al tercero, y %9 al que originalmente estaba colocado en la décima posición. Una nueva

ejecución de SHIFT desplaza nuevamente los parámetros en la misma forma, mandando el tercer parámetro a %1, el cuarto a %2, el onceavo a %9,etc. Esto puede observarse en el siguiente diagrama.

Se ejecuta:	PROG	ESTOS	SON	LOS	PARAMETROS	
Al iniciar PROG:	%0	%1	%2	%3	%4	%5
Ejecutá SHIFT:		%0	%1	%2	%3	%4
Ejecuta SHIFT:			%0	%1	%2	%3
Ejecuta SHIFT:				%0	%1	%2
Ejecuta SHIFT:					%0	%1

Como puede verse, este comando permite procesar mas de nueve parámetros dentro del programa BATCH, sin embargo, la verdadera importancia del comando SHIFT radica en el hecho de que permite procesar todos los parámetros proporcionados al programa en una forma homogénea mediante un cierto proceso repetitivo sobre el parámetro indicado por %1, el cual debe terminar ejecutando un comando SHIFT y preguntando si aún existe otro parámetro, en cuyo caso se regresará a procesarlo con la misma sección de programa ya utilizada. A este respecto considere los comandos siguientes:

```
:REGRESA
PROCESA %1
SHIFT
IF NOT "%1" == "" GOTO REGRESA
```

Al comenzar el programa se procesará el parámetro colocado en primer lugar representado por %1; en seguida, el comando SHIFT colocará el siguiente parámetro en %1, hecho lo cual el comando IF preguntará si aún queda otro parámetro por procesar, regresando a hacerlo en caso afirmativo.

Note que la forma de detectar el fin de los parámetros es cuando uno de ellos ya no existe, es decir, cuando se sustituye por "nada"; sin embargo, el comando IF siempre debe estar escrito correctamente, y puesto que no hay forma de escribir "nada", es necesario añadir cualquier caracter a ambos lados de los signos de igual con el fin de que el comando IF se ejecute correctamente sin marcar errores. El caracter añadido puede ser cualquiera; por ejemplo, si se incluyera la letra X tendríamos lo siguiente:

```
IF NOT %1X == X GOTO REGRESA
```

Este comando IF tendría el mismo efecto que el antes mostrado; sin embargo el colocar dos comillas inmediatamente juntas para indicar que hay "nada" es una forma establecida en otros lenguajes de programación, por lo que convendrá conservarla.

A continuación se muestra la forma estándar de procesar una serie de parámetros en un programa BATCH utilizando el comando SHIFT.

Programa de ejemplo: EJ-SHIFT.BAT

```
ECHO OFF
CLS
:OTRO-PAR
REM EN ESTE PUNTO VA EL PROCESO DESEADO SOBRE EL PARAMETRO %1
REM EN ESTE CASO SE LIMITA A MOSTRAR EL PARAMETRO EN LA PANTALLA
ECHO PARAMETRO PROCESADO: %1
REM LOS SIGUIENTES COMANDOS DESPLAZAN AL SIGUIENTE PARAMETRO
REM Y REPITEN EL PROCESO SI AUN QUEDA ALGUNO POR PROCESAR
SHIFT
IF NOT "%1" == "" GOTO OTRO-PAR
REM FIN DEL PROCESO
```

El programa de ejemplo anterior se puede modificar con el fin de realizar cualquier proceso deseado sobre una serie de archivos, por ejemplo copiá o borrado, proporcionados en los parámetros de un programa BATCH.

Finalmente, es importante mencionar que una vez ejecutado algún comando SHIFT y desplazado la posición de los parámetros, no habrá forma de regresar al parámetro que se haya dejado atrás.

2.6 Comando FOR

Forma de uso: FOR %%c IN (lista de palabras) DO comando
comodín

Ejecuta un comando varias veces.

El comando FOR permite ejecutar varias veces otro comando con la caracteristica de poder modificar alguno de sus parámetros en cada ejecución, con el fin de que el comando se ejecute realmente sobre un conjunto de datos diferentes.

La parte indicada por %%C debe ser un par de caracteres tanto por ciento seguidos por una letra cualquiera; estos tres caracteres se podrán incluir en la parte marcada COMANDO para indicar el parámetro que variará con cada ejecución del mismo.

El comando FOR tiene dos formas de uso dependiendo de lo que se coloque dentro de los paréntesis. En caso de colocar una LISTA DE PALABRAS, el comando FOR sustituirá los caracteres %%C por cada una de las palabras incluídas en la lista y por cada sustitución hecha ejecutará al COMANDO. Esto implica que si el COMANDO incluye los caracteres %%C en alguno de sus parámetros, el COMANDO se ejecutará una vez con cada una de las palabras dadas.

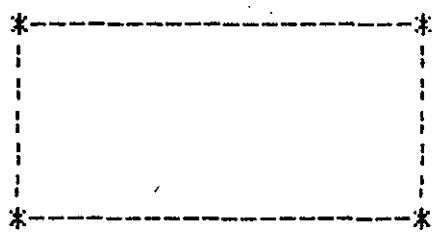
```
FOR %%F IN (ARCH.DAT PROG.FOR EJEM.TXT) DO COPY %%F B:
```

El comando FOR anterior copiaría los tres archivos indicados al disco de la unidad B:. Si los caracteres %%C no se incluyen en el COMANDO, éste se ejecutará varias veces en forma idéntica, lo cual puede ser útil en algunos casos como el mostrado en seguida.

Programa de ejemplo: EJ-FOR1.BAT

```
ECHO OFF
CLS
REM EL SIGUIENTE COMANDO TERMINA CON LA TECLA F7:
FOR %%I IN (1 2 3 4) DO ECHO ^@
ECHO *--------------------*
FOR %%I IN (1 2 3 4 5) DO ECHO ¦                    ¦
ECHO *--------------------*
```

El programa BATCH anterior mostraría lo siguiente en la pantalla, precedido por cuatro líneas en blanco:



Las palabras colocadas dentro de los paréntesis deberán estar separadas entre sí por un espacio en blanco o por alguno de los siguientes caracteres: coma (,), punto y coma (;), o igual (=).

Si lo que se coloca dentro de los paréntesis es un nombre de archivo que incluya un COMODIN (descrito en el manual de MS-DOS con el término "wild-card"), entonces el comando FOR sustituirá los caracteres %%C por el nombre de cada uno de los archivos que existan en el disco seleccionados por el comodín, y por cada sustitución hecha ejecutará al COMANDO. Esto permite ejecutar sobre varios archivos aquellos comandos de MS-DOS que no admiten comodines, como en el ejemplo mostrado a continuación.

Programa de ejemplo: EJ-FOR2.BAT

```
FOR %%F IN (%1) DO TYPE %%F
```

El programa anterior, que consta de una sola línea, permitirá mostrar en la pantalla varios archivos seleccionados por el comodín indicado por el primer parámetro. Por ejemplo, para observar el contenido de todos los programas BATCH del disco:

```
A> EJ-FOR2 *.BAT
```

A los tres caracteres %%C se les puede bautizar con el término "variable temporal" del comando FOR, ya que el valor de ellos cambia con cada ejecución del comando y además ese valor

solamente existe mientras se ejecuta el comando FOR. Este término será útil posteriormente, cuando se hable de la variable temporal de otros comandos similares al comando FOR en este sentido.

Un pequeño inconveniente del comando FOR es el hecho de que la letra C que forma la variable temporal junto con los caracteres %% colocados en la parte del COMANDO deberá ser exactamente la misma que la colocada enseguida de la palabra FOR, incluídos los tamaños de ambas letras, por lo que si por un error de tecleo los primeros %%C tienen una letra mayúscula y los segundos %%c la tienen minúscula, éstos tres caracteres no serán sustituídos.

El comando FOR junto con el comando IF son los dos comandos de MS-DOS que se forman con dos partes: una es la que le corresponde al comando FOR o IF por sí mismos, y la otra la que corresponde a cualquier otro comando de MS-DOS que se coloque enseguida del FOR o el IF. Esto abre la posibilidad de combinar estos dos comandos entre sí, para lo cual hay que seguir las siguientes reglas.

Es posible combinar un IF dentro de otro IF con el fin de establecer condiciones múltiples; en este caso el comando colocado después del segundo IF solo se ejecutará cuando las condiciones impuestas por ambos IF's se cumplan; por ejemplo:

```
IF %1 == COPIA IF NOT %2 == PRIVADO.DAT COPY %2 B:
```

El comando anterior copiará el archivo indicado por el segundo parámetro al disco de la unidad B cuando el primer parámetro sea la palabra COPIA, excepto si el archivo es PRIVADO.DAT.

En forma similar se puede combinar un IF dentro de un FOR, o un FOR dentro de un IF. Por ejemplo, el siguiente comando borrará todos los archivos cuyo nombre esté dado por el primer parámetro y que tengan cualquier extensión, excepto aquél cuya extensión sea precisamente la indicada por el segundo parámetro:

```
FOR %%F IN (%1.*) DO IF NOT %%F == %1.%2 DEL %%F
```

La única combinación prohibida en BATCH es incluir un FOR dentro de otro FOR, por lo que no se podrá hacer lo que se llama un "anidamiento" de comandos FOR. Otra posibilidad, que si bien no está prohibida será recomendable evitar siempre, es el incluir un comando GOTO dentro de un FOR. Lo que se obtiene de esta combinación se puede efectuar de otra manera, según se indica en los ejemplos incluídos en la descripción del comando SET.

Capítulo 3 - MANEJO DE VARIABLES BATCH

3.1 Concepto de variable

El manejo de variables es un tema familiar para cualquier persona que conozca un lenguaje de programación, ya que las variables son un medio que permite ampliar las capacidades de cualquier lenguaje permitiendo hacer ciertas manipulaciones que no sería posible llevar a cabo sin el apoyo de una variable.

En sí, una variable es un elemento del lenguaje capaz de almacenar un valor con el fin de ser utilizado posteriormente; de esta manera, las variables pueden "recordar" valores para ser utilizados una y otra vez sin tener que volverlos a generar en cada ocasión. El valor almacenado en una variable puede ser utilizado en diversas formas, por ejemplo, para combinarse entre ellos con el fin de generar un nuevo valor, para cambiar de valor en base a un valor anterior, como sería contar las veces que se ha ejecutado un cierto proceso y muchas otras posibilidades mas.

El lenguaje de programación BATCH, al igual que el resto de los lenguajes de programación, tiene la capacidad de manejar variables. Las variables BATCH se definen mediante el comando interno SET de MS-DOS en la forma descrita a continuación.

3.2 Comando SET

Forma de uso: SET variable=secuencia

Asigna una secuencia de caracteres a una variable.

El comando SET de MS-DOS permite definir en cualquier momento una variable BATCH, esto es, permite crear una variable y almacenar un valor en su interior. La definición de variables en BATCH es similar a la del lenguaje BASIC en algunos aspectos; a continuación se dan las reglas completas a este respecto.

Para asignar un valor a una variable BATCH se utiliza el comando SET de acuerdo a la forma de uso mostrada anteriormente, p.e.:

```
SET EJEM=VALOR DE LA VARIABLE
```

El comando anterior asignaría a la variable "EJEM" la secuencia de caracteres "VALOR DE LA VARIABLE". Nótese que en esta descripción se utilizarán las comillas solamente para marcar claramente el nombre de la variable y el valor de la misma, y que las comillas no se requieren en el comando SET; si se colocaran, éstas se incluirían en la secuencia de caracteres asignada a la variable. Note también que en este manual se aplicará indistintamente el término "valor" o "secuencia de caracteres" al conjunto de caracteres almacenados dentro de una variable BATCH.

Las variables BATCH solamente pueden almacenar secuencias de caracteres, por lo que no hay variables BATCH numéricas propiamente dichas. De acuerdo con esto, el siguiente comando:

```
SET NUMERO=12345
```

asignaría a la variable "NUMERO" la secuencia de caracteres "12345". El valor asignado a una variable puede incluir cualquier caracter excepto un signo de igual (=), por lo que el siguiente comando es inválido y marcaría un error en caso de ejecutarse:

```
SET MENSAJE=RESULTADO = 3.5
```

A diferencia de otros lenguajes de programación, los espacios en blanco sí afectan al comando SET, de tal manera que si se inserta algún espacio en blanco después del nombre de la variable o dentro de su valor, tales espacios serán incluidos dentro de esas partes, como es el caso del segundo comando siguiente:

```
SET NOMBRE=BATCH
SET NOMBRE = BATCH
```

El primer comando asignaría a la variable "NOMBRE" el valor "BATCH", mientras que el segundo asignaría a la variable "NOMBRE " (que es distinta de "NOMBRE") la secuencia de caracteres " BATCH" (que es distinta de "BATCH"), es decir, el nombre de la variable será todo lo que esté colocado a la izquierda del signo igual, y su valor todo lo que esté a la derecha del mismo signo. Debido a esta característica es conveniente adoptar la costumbre de no dejar nunca espacios en blanco junto al signo (=) de los comandos SET que utilicemos.

La única facilidad que proporciona el comando SET en este sentido es la de convertir el nombre de una variable que se haya dado en letras minúsculas a letras mayúsculas, de tal manera que ambas formas de escribir el nombre indiquen la misma variable, p.e.

```
set letrero=letras minúsculas
```

El comando SET anterior asignaría a la variable "LETRERO" la secuencia "letras minúsculas". Nótese que el valor asignado no se convierte a mayúsculas, solamente el nombre en sí de la variable.

Por otro lado, y también en forma diferente a los demás lenguajes, el nombre de una variable BATCH puede contener cualquier caracter que sea válido insertar desde el teclado, e inclusive no es forzoso que éste empiece con una letra. Esto ya se dejó entrever anteriormente cuando se indicó que todos los caracteres colocados antes del signo igual forman parte del nombre, inclusive si se trata de espacios en blanco.

Esta característica puede ser un inconveniente o una ventaja, dependiendo de cómo sea utilizada. A continuación se muestra un par de ejemplos de cada uno de los casos mencionados:

```
SET 36=VALOR DE LA VARIABLE "36"

SET SOY UNA VARIABLE=ESTE ES MI VALOR

SET VECTOR[1]=¿ELEMENTO SUBINDICE 1?

SET MATRIZ[2,3]=¿ELEMENTO SUBINDICES 2,3?
```

Los dos primeros casos son ejemplos del uso inadecuado de esta facilidad: en el primero se trata de una variable cuyo nombre no empieza con una letra, y en el segundo de una variable cuyo nombre incluye espacios en blanco. Estas dos formas de nombres de variables deberán evitarse siempre; en todo caso, si se desean separar las palabras dentro del nombre de una variable inclúyase el caracter underscore (_), por ejemplo: SOY_UNA_VARIABLE.

Sin embargo, los dos últimos casos, en los cuales se incluyen los caracteres extraños "[1]" y "[2,3]" dentro del nombre de la variable pueden ser muy interesantes, ya que precisamente estos caracteres "raros" escritos de esta manera nos permitirán simular el manejo de vectores y matrices en el lenguaje BATCH, lo cual puede proporcionar un gran poder de programación.

Cuando se utilicen variables BATCH en un programa hay que tener presente que el espacio disponible para definir variables tiene un cierto tamaño fijo que sólo permitirá definir un cierto número de variables, por lo que será recomendable que al terminar cada programa BATCH se borren todas las variables que el programa haya utilizado. Por otro lado, si las variables no fueran borradas posteriormente no se sabría cuál programa las definió ni para qué fueron utilizadas, por lo que se desperdiciarían. Para borrar una variable BATCH se le asigna un valor nulo de la siguiente forma:

```
SET VARIABLE_A_BORRAR=
```

Finalmente, si se ejecuta el comando SET sin parámetros se mostrarán en la pantalla el nombre y el valor de todas las variables BATCH definidas hasta ese momento. Hay que tener presente que puede haber hasta tres variables que definió y utiliza el propio sistema operativo MS-DOS, llamadas "COMSPEC", "PATH" y "PROMPT", que sólo podrán ser modificadas siguiendo las indicaciones al respecto descritas en el manual de MS-DOS.

3.3 Utilización de variables BATCH

Una vez que se ha definido una variable BATCH el paso lógico siguiente es utilizar su valor. Puesto que las instrucciones del lenguaje BATCH están formadas por comandos de MS-DOS, debe ser obvio que el valor de las variables BATCH será utilizado para completar algunos de estos comandos, proporcionando algún parámetro o alguna parte de un comando de MS-DOS.

Para utilizar el valor de una variable BATCH dentro de un archivo de comandos BATCH se debe encerrar el nombre de la variable deseada entre un par de caracteres tanto por ciento, p.e.:

```
SET UNO=VALOR DE LA VARIABLE
SET DOS=%UNO%
```

El segundo comando SET anterior asignaría a la variable "DOS" el mismo valor previamente asignado a la variable "UNO", el cual es "VALOR DE LA VARIABLE". Hay que notar que esta sustitución del nombre de una variable encerrado entre porcientos por su valor sólo ocurre dentro de un archivo de comandos BATCH; si se ejecuta directamente desde el teclado no se hará ninguna sustitución.

Una ventaja importante que tiene BATCH sobre los demás lenguajes de programación es el hecho de que una variable BATCH puede ser utilizada de esta manera en cualquier parte dentro de un programa BATCH, lo cual implica que una variable BATCH puede proporcionar un parámetro a un comando, varios o todos los parámetros de un comando e inclusive un comando completo, es decir, no importa en donde se coloque una variable BATCH encerrada entre porcientos en un programa: cuando éste se ejecute se colocará en ese lugar el valor de la variable y el texto que quede será ejecutado.

Ejemplos individuales:

```
ECHO %MENSAJE%
```
Muestra en la pantalla el valor de la variable "MENSAJE".

```
SET TODO_JUNTO=%UNO% %DOS%-%TRES%
```
Concatena los valores de las variables "UNO", "DOS" y "TRES", separando el valor de "UNO" y "DOS" por un espacio en blanco, y el valor de "DOS" y "TRES" por un guión, y el resultado de este agrupamiento lo asigna a la variable "TODO_JUNTO".

```
GOTO %ETIQUETA%
```
Transfiere el control a la etiqueta que haya sido asignada previamente a la variable "ETIQUETA".

```
COPY %QUE_COPIO%
```
Copia los archivos indicados por la variable "QUE_COPIO", la cual puede indicar una multitud de tipos de copia diferentes. Si previamente se hubiera ejecutado: SET QUE_COPIO=*.TXT C: entonces el comando anterior copiaría los archivos de texto al disco fijo.

```
%LO_QUE_SEA%
```
Si la línea anterior se incluye en un programa BATCH, ejecutará el comando completo que previamente hubiera sido asignado a la variable "LO_QUE_SEA". Por ejemplo, si previamente se hubiera ejecutado lo siguiente: SET LO_QUE_SEA=TYPE ARCH.TXT la línea anterior mostraría en la pantalla el archivo ARCH.TXT. Como puede verse fácilmente, esta capacidad tiene alcances ilimitados ya que es capaz de ejecutar practicamente cualquier proceso.

A continuación se describen tres maneras diferentes de aprovechar las variables BATCH, las cuales ayudarán a entender con mayor claridad las posibilidades a las que se tiene acceso cuando se utilizan las variables dentro de un programa BATCH.

3.4 Primer ejemplo: Agrupamiento de parámetros

Este primer ejemplo de uso de variables describe la forma de agrupar todos los parámetros de un programa BATCH en una sola variable, y los beneficios que se puede obtener de ello.

Como ya se explicó anteriormente, cuando un programa BATCH se ejecuta se tiene acceso a los parámetros con que fué activado, lo cual se hace colocando un caracter tanto por ciento seguido por un dígito, pudiendo procesar todos los parámetros proporcionados mediante un ciclo repetitivo con el apoyo del comando SHIFT.

Sin embargo, existen numerosas ocasiones en las que es deseable que dentro de un programa BATCH se tomen todos sus parámetros de una sola vez en la misma forma en que éstos fueron proporcionados y sin importar cuántos sean, en lugar de tomarlos individualmente mediante un %1, %2, %3, etc., o en un ciclo repetitivo.

Un ejemplo de este caso sería cuando se utiliza un programa BATCH para grabar en un archivo de disco algún mensaje proporcionado en los parámetros. Si cada parámetro se graba en el archivo con un comando ECHO mediante un proceso repetitivo que incluya un SHIFT, se grabará una sola palabra por cada línea del archivo, mientras que si se graban los parámetros desde el %1 hasta el %9 en un solo comando ECHO y después se ejecutan nueve SHIFT's, el mensaje se grabará siempre en líneas de nueve palabras o menos.

Lo que realmente se desea es agrupar todos los parámetros proporcionados al programa BATCH antes de proceder a procesarlos, es decir, se necesita contar con un lugar en el cual se pueda almacenar el primer parámetro, después añadirle el segundo, el tercero, y así sucesivamente hasta que en ese lugar se encuentren almacenados todos los parámetros proporcionados al programa. Una vez hecho esto, bastará con tomar ese valor almacenado con el fin de procesar todos los parámetros proporcionados de una sola vez.

Al recordar el funcionamiento de las variables BATCH nos damos cuenta de que ésta es precisamente la labor para la cual fueron diseñadas. Para resolver este problema bastará con que a una cierta variable le asignemos el valor del primer parámetro mediante un comando SET y después se haga el siguiente ciclo repetitivo: se pase al siguiente parámetro con un comando SHIFT, se tome este parámetro y se añada al valor anterior de la variable, y se almacene esta agrupación en la misma variable con el fin de que quede lista para el ciclo siguiente. La agrupación de los dos valores y su asignación a la misma variable se hace con un sólo comando SET de acuerdo al siguiente ejemplo:

```
SET VARIABLE=%VARIABLE% %1
```

La primera vez que se ejecute estè comando se tomará el primer parámetro, almacenado en la variable al principio del proceso, mas el segundo, que se colocó en %1 con el primer comando SHIFT, y esta agrupación se almacenará en la misma variable; la segunda vez se añadirán a los dos parámetros anteriores el tercero, y así sucesivamente. Los parámetros estarán separados uno de otro por un espacio en blanco, el mismo que está colocado entre ambos valores en el comando SET anterior. Este ciclo deberá repetirse hasta haber agrupado todos los parámetros existentes.

A continuación se muestra el proceso completo de agrupación de parámetros. En este proceso, así como en todos los programas de ejemplo que lo incluyen, se utilizará el nombre "ARGV" para la variable que agrupa los parámetros, debido a que es el nombre utilizado en otro lenguaje de programación para este mismo fin.

```
REM SE ASIGNA EL PRIMER PARAMETRO A "ARGV"
SET ARGV=%1
:OTROPAR
REM SE PASA AL SIGUIENTE PARAMETRO
SHIFT
REM SI YA NO HAY OTRO PARAMETRO: SE TERMINA
IF "%1" == "" GOTO FINPAR
REM DE LO CONTRARIO: SE AÑADE EL PARAMETRO A LOS ANTERIORES
SET ARGV=%ARGV% %1
REM Y SE REPITE EL PROCESO ANTERIOR
GOTO OTROPAR
:FINPAR
```

Un ejemplo del uso de la agrupación de los parámetros se muestra a continuación. El segundo programa de ejemplo incluído en la descripción del comando FOR está encargado de mostrar en la pantalla los archivos seleccionados por el comodín dado por su primer parámetro. Ese programa de ejemplo se llama EJ-FOR2.BAT y está formado solamente por el siguiente comando:

```
FOR %%F IN (%1) DO TYPE %%F
```

Con el fin de facilitar el uso de este programa se le podría añadir la capacidad de que incluyera todos los parámetros proporcionados en la lista de palabras a procesar por el comando FOR, de manera que si se ejecutara con el nombre de tres archivos, mostrara el contenido de los tres archivos dados, mientras que si se ejecutara con un comodín, mostrara el contenido de todos los archivos seleccionados por el comodín. A continuación se presenta esta nueva versión de ese programa.

Programa de ejemplo: EJ-SET1.BAT

```
ECHO OFF
SET ARGV=%1
:OTROPAR
SHIFT
IF "%1" == "" GOTO FINPAR
SET ARGV=%ARGV% %1
GOTO OTROPAR
:FINPAR
FOR %%F IN (%ARGV%) DO TYPE %%F
SET ARGV=
```

Nótese que el último comando del programa anterior está encargado de borrar la variable "ARGV" antes de terminar el programa. Si se deseara ver el contenido de todos los programas BATCH del disco, bastaría con ejecutar la siguiente línea:

```
A> EJ-SET1 *.BAT
```

Mientras que para ver solamente el contenido de los archivos UNO.BAT, DOS.BAT y TRES.BAT, se usaría de esta forma:

```
A> EJ-SET1 UNO.BAT DOS.BAT TRES.BAT
```

Esta forma de utilizar un programa BATCH es muy cómoda, ya que si se desean procesar ciertos archivos, se indican éstos, y si se desean procesar varios archivos seleccionados por un comodín, basta con poner el comodín; sin embargo, hay que tener presente que el comando FOR de la versión 2 de MS-DOS solamente podrá procesar un solo comodín y no una serie de ellos, por lo que sería interesante desarrollar un programa BATCH que pudiera procesar de igual forma una lista de palabras o de comodines, sin importar la versión de MS-DOS que se utilice, y por cada archivo seleccionado ejecutara con él un cierto comando cualquiera.

3.5 Segundo ejemplo: Identificación de una opción

En este segundo ejemplo se describe otro posible uso de las variables BATCH para facilitar la identificación de una cierta opción proporcionada a un programa BATCH. Esta descripción es interesante pues no se limita a mostrar la forma de utilizar las variables en un programa, sino que analiza una serie de características de operación que deben considerarse cuando se utiliza al lenguaje BATCH en forma avanzada. Se recomienda leer de una sola vez esta descripción con el fin de no dejar truncos los nuevos conceptos involucrados en ella.

Un uso frecuente del lenguaje de programación BATCH es el desarrollo de algunos programas capaces de efectuar tres o cuatro procesos diferentes, pero relacionados entre sí. Para seleccionar el proceso deseado usualmente se proporciona una palabra clave en un parámetro, la cual indica el proceso particular a realizar.

Por ejemplo, suponga un programa BATCH encargado de mantener en un archivo de disco una serie de apuntes o notas breves y cuya finalidad es la de desechar las hojas de papel y el lápiz que se pudieran utilizar para esta misma labor. Este programa, que se podría llamar NOTAS.BAT, necesitaría que mediante su primer parámetro se le indicara el proceso a realizar de entre sus posibles opciones, las cuales pudieran ser las siguientes:

```
INSERTA: inserta una nueva nota en el archivo.
MUESTRA: muestra el archivo de notas en la pantalla.
IMPRIME: imprime el archivo de notas.
BORRA:   borra el contenido del archivo de notas.
```

Con estas opciones, las diversas ejecuciones del programa NOTAS.BAT podrían tener la siguiente forma:

```
A> NOTAS INSERTA NUEVA NOTA A INSERTAR

A> NOTAS MUESTRA

A> NOTAS IMPRIME

A> NOTAS BORRA
```

Al analizar este programa se observa de inmediato que hay que comenzar identificando la palabra proporcionada por el primer parámetro con el fin de saber cuál proceso se debe realizar de entre los cuatro posibles. Esto se puede evitar mediante un truco "sucio" mostrado en algunos ejemplos de programas BATCH incluídos en ciertos libros, que consiste en no identificar la palabra en lo absoluto sino simplemente ejecutar un comando GOTO hacia la etiqueta indicada por el primer parámetro, es decir:

```
GOTO %1
```

De esta manera, bastaría con incluir dentro del programa BATCH cuatro etiquetas correspondientes a cada una de las opciones para que el comando anterior funcionara correctamente. A pesar de que ésta es una solución muy sencilla y corta en términos de programación, tiene el grave inconveniente de que si el usuario teclea cualquier otra palabra que no sea alguna de las opciones definidas, el programa marcaría el error "Label not found" en la ejecución del comando GOTO, puesto que tal etiqueta no existe.

Todos los sistemas de aplicación deben desarrollarse pensando que el usuario eventualmente se equivocará, y dentro de lo posible deben ser lo suficientemente amistosos como para indicarle al usurio en qué se equivocó y qué debe hacer para corregir su error. Puesto que es probable que los programas desarrollados en BATCH los utilice un mayor número de usuarios que cualquier otro paquete de aplicación, ya que BATCH no está orientado hacia un

fin particular sino que se refiere a la utilización cotidiana de la computadora en sí, esta característica de "amabilidad" de los sistemas que se desarrollen con BATCH es de suma importancia.

Por lo tanto, el sencillo truco anterior no es lo mas indicado cuando se desean desarrollar programas BATCH de un alto nivel similares a éste. Una mejor solución sería el comparar la palabra proporcionada contra las opciones definidas, y en caso de no ser ninguna de ellas que se mostrara una breve lista de las opciones existentes y de su finalidad. La identificación de la opción debe hacerse con el comando IF, pero debido a la limitación propia de este comando de no conjuntar letras mayúsculas con minúsculas, las comparaciones deberán ser hechas dos veces: una con la opción escrita con letras mayúsculas y la otra con minúsculas, ya que el usuario puede emplear cualquier tamaño al utilizar el programa.

A continuación se muestra la sección del programa encargada de identificar la palabra proporcionada en el primer parámetro y mostrar una breve descripción en caso de que ésta sea errónea, o bien, de transferir el control a la etiqueta encargada de ejecutar el proceso respectivo. Nótese que la descripción de ayuda también se muestra si no se proporciona ningún parámetro, lo que permite contar con una pequeña ayuda inmediata de la forma de utilizar el programa. Esta última característica constituye un estándar en muchos programas de MS-DOS y adicionales que convendrá conservar en todos los programas que desarrollemos.

```
ECHO OFF
IF "%1" == "" GOTO AYUDALE
IF %1 == INSERTA GOTO INSERTA
IF %1 == inserta GOTO INSERTA
IF %1 == MUESTRA GOTO MUESTRA
IF %1 == muestra GOTO MUESTRA
IF %1 == IMPRIME GOTO IMPRIME
IF %1 == imprime GOTO IMPRIME
IF %1 == BORRA GOTO BORRA
IF %1 == borra GOTO BORRA
:AYUDALE
ECHO La forma de usar este programa es la siguiente:
ECHO
ECHO NOTAS opción nota
ECHO
ECHO En donde la "opción" debe ser alguna de las siguientes:
ECHO INSERTA  inserta una nueva nota en el archivo
ECHO MUESTRA  muestra el archivo de notas en la pantalla
ECHO IMPRIME  imprime el archivo de notas
ECHO BORRA    borra el contenido del archivo de notas
GOTO FIN
:INSERTA
REM EN ESTE PUNTO VA EL PROCESO DE INSERCION
GOTO FIN
:MUESTRA
REM EN ESTE PUNTO VA EL PROCESO DE MOSTRADO
GOTO FIN
```

```
:IMPRESION
REM EN ESTE PUNTO VA EL PROCESO DE IMPRESION
GOTO FIN
:BORRA
REM EN ESTE PUNTO VA EL PROCESO DE BORRADO
:FIN
```

El programa anterior ya proporciona el grado de amabilidad recomendado, sin embargo, también presenta varios inconvenientes que mencionaremos enseguida. Como puede verse facilmente, el método utilizado para identificar una determinada opción es bastante rudimentario además de ineficiente. Una de las partes que entorpecen la ejecución de BATCH en mayor grado es la lectura en sí de los comandos, por lo que mientras mas comandos sea necesario revisar para realizar una determinada acción, mas tiempo se llevará en realizarse ésta.

Si la opción proporcionada al programa de ejemplo anterior fuera "borra", para que el programa la identifique debe ejecutar antes nueve comandos IF, mientras que si fuera "INSERTA", sólo debe ejecutar dos. Esto puede significar una diferencia en tiempos de ejecución entre la primera y la última opciones de la lista que puede llegar a ser notable. Por otro lado, en este ejemplo sólo se tienen 4 opciones diferentes; si un programa BATCH tuviera 8 o 9 opciones el tiempo utilizado en la identificación de la última opción de la lista puede llegar a ser intolerable.

Al utilizar BATCH siempre se debe buscar que los programas sean lo mas corto posible, es decir, que se obtenga el mayor provecho posible de cada una de las líneas incluídas en un programa BATCH. Ahora bien, el problema que estamos tratando de resolver consiste en comparar un parámetro proporcionado a un programa BATCH contra una lista de opciones previamente establecidas, por lo que antes de tratar de resolver este problema debemos reflexionar en la siguiente pregunta: ¿Existe algún comando de MS-DOS que nos permita comparar un parámetro contra una lista de palabras?

La respuesta a esta pregunta es: ¡SI!. Si bien no tenemos un comando de MS-DOS que directamente compare un parámetro contra una lista de palabras, tenemos en cambio un comando que puede repetir un proceso similar sobre una lista de palabras, el cual es el comando FOR, y otro comando que puede comparar una palabra contra otra, que es el comando IF; si combinamos estos dos comandos entre sí obtendremos el proceso que necesitamos.

En el desarrollo de programas BATCH grandes siempre se debe buscar una posible combinación en una sola línea de dos o mas comandos que realicen la misma labor que dos o mas comandos individuales, ya que al estar los comandos combinados en una sola línea de un programa BATCH su lectura será mas rápida que la de los comandos separados. Por otro lado, si combinamos un comando FOR con cualquier otro comando, la ejecución de éste será mucho mas rápida que si colocamos varias veces el comando individual.

Por supuesto, los mejores candidatos para hacer combinaciones son los comandos FOR e IF, ya que precisamente están diseñados para ello; sin embargo, algunos paquetes de aplicación pueden incluir ciertos comandos diseñados con esta característica, los cuales también permitirán algunas combinaciones interesantes.

Regresando a nuestro problema, para comparar el primer parámetro contra las cuatro opciones establecidas en nuestro programa de ejemplo bastaría con ejecutar el siguiente comando FOR:

```
FOR %%O IN (INSERTA inserta MUESTRA muestra IMPRIME imprime
            BORRA borra) DO IF %1 == %%O . . .
```

Si el comando anterior parece muy largo, hay que recordar que se cuenta con un máximo de 127 caracteres para cada una de las líneas incluídas en un programa BATCH, lo cual quiere decir que al comando anterior todavía le faltan 38 caracteres para llegar a este límite. Este comando se muestra en este texto en dos líneas por razones de espacio, pero por supuesto que al incluirse dentro de un programa BATCH se debe teclear en una sola línea. Solo falta definir el comando que debe colocarse enseguida del comando IF, para lo cual hay que hacer un breve análisis.

El comando FOR sustituirá los caracteres %%O por cada una de las palabras incluídas dentro de los paréntesis, y por cada una de las sustituciones hechas ejecutará al comando IF; el comando IF a su vez comparará cada una de estas palabras con el primer parámetro proporcionado al programa. Ahora bien, cuando ambas palabras sean iguales querrá decir que ya se identificó al parámetro dado, por lo tanto, ¿que debe hacerse en este caso?.

Una primera idea nos podría sugerir que en este caso hay que transferir el control a la etiqueta indicada por el parámetro; puesto que el parámetro ya fué igual a alguna de las opciones, el control se transferirá a una etiqueta que efectivamente exista. Por otro lado, si el parámetro no fué ninguna de esas palabras no se ejecutará nunca la transferencia, por lo que en este caso la ejecución continuaría con la línea siguiente al comando FOR, la cual entonces deberá mostrar las descripciones de ayuda. Veamos:

```
ECHO OFF
IF "%1" == "" GOTO AYUDALE
FOR %%O IN (INSERTA inserta MUESTRA muestra IMPRIME imprime
            BORRA borra) DO IF %1 == %%O GOTO %1
:AYUDALE
```

El resto del programa sería igual al ejemplo mostrado anteriormente. Al parecer el problema ya está resuelto; sin embargo, si analizamos el comando FOR anterior encontraremos un conflicto en su operación, que se describe en seguida.

El comando FOR indica ejecutar varias veces al comando colocado enseguida de él con todas y cada una de las palabras incluídas dentro de los paréntesis, mientras que el comando GOTO indica

interrumpir este proceso en algún momento y transferir el control a otra línea del programa BATCH, lo cual implica que se quedaría pendiente la comparación del resto de las palabras por lo que la operación del FOR y del GOTO se contrapone la una a la otra.

Cuando utilicemos el lenguaje BATCH en forma avanzada hay que recordar que de ninguna manera se le puede tratar como si fuera un lenguaje de programación común, p.e. BASIC, ya que por un lado no fué diseñado para ello, y por otro los manuales de MS-DOS y todos los libros "avanzados" relacionados con este tema no dan ninguna información referente a cuestiones de este tipo.

Si intentamos resolver esta clase de problemas haciendo pruebas de la forma en que se comporta BATCH y actuando en consecuencia, es muy probable que nuestros programas no funcionen correctamente en versiones posteriores de MS-DOS. No podemos confiar en que diferentes versiones de MS-DOS se comporten de igual manera en la ejecución de detalles que ni siquiera están mencionados en el manual; la probabilidad de que estos detalles cambien es alta.

Por lo tanto siempre que se presente algún conflicto de este tipo lo que se debe hacer es evitar esa situación y hacer el proceso en forma diferente. Regresando a nuestro ya viejo problema de la identificación de la opción dada en el primer parámetro, ¿lo anterior significa que no podremos utilizar el FOR con el IF anidado que en forma tan elegante había resuelto el problema?. No precisamente; en realidad el problema no estriba en el FOR ni en el IF, sino en el GOTO incluído dentro de ambos, por lo que bastará con eliminar al comando GOTO para eliminar el problema.

Ahora bien, si no se utiliza el comando GOTO anidado dentro del IF, ¿de qué servirá que un comando FOR ejecute 8 veces un comando IF si de todas maneras debe continuar con el comando colocado después de él?. Para poderse responder, esta pregunta debe hacerse de la siguiente manera: ¿cómo se puede obtener provecho de un comando FOR que se ejecutó en forma independiente con anterioridad?. La respuesta a esta pregunta sí la conocemos.

Es muy probable que a estas alturas el lector ya haya olvidado la razón que nos sumergió en este problema, por lo que convendrá recordarla: se trata de un ejemplo de uso de las variables BATCH.

Efectivamente, la idea consiste en obtener provecho de un comando FOR después de que este fué ejecutado, y la única forma de hacerlo es "recordando" algún resultado intermedio generado durante la ejecución del comando FOR con el fin de ser utilizado posteriormente. Por supuesto, la única manera de almacenar resultados para su uso posterior es mediante una variable BATCH.

Por lo tanto, para resolver por fin de una manera correcta este problema bastará con que una vez que identifiquemos la palabra que se proporcionó en el primer parámetro almacenemos este resultado en una variable BATCH. El valor almacenado debe ser el mismo valor del parámetro, de tal forma que al terminar el ciclo

del comando FOR el control se pueda transferir a la etiqueta ahora almacenada en esa variable. Por ejemplo, si utilizáramos la variable "ETIQUETA" con este fin, tendríamos lo siguiente:

```
FOR %%O IN (INSERTA inserta MUESTRA muestra IMPRIME imprime
            BORRA borra) DO IF %1 == %%O SET ETIQUETA=%1
GOTO %ETIQUETA%
```

Sin embargo, ¿que sucederá si el parámetro proporcionado no es ninguna de las palabras definidas?. En este caso no se asignará ningún valor a la variable "ETIQUETA" dentro del FOR, pero de todas maneras la instrucción que sigue del FOR es el GOTO que utiliza el valor de esa variable. Por lo tanto, antes de ejecutar la instrucción FOR se debe almacenar una etiqueta inicial dentro de esa variable, la cual debe ser la etiqueta a la que se desee transferir el control en el caso de que el parámetro proporcionado no sea ninguna de las opciones definidas.

Una asignación de un valor a una variable previa a la ejecución de un proceso que puede modificarla se denomina "inicialización" de esa variable. Por lo tanto, finalmente basta con inicializar la variable "ETIQUETA" con la etiqueta "AYUDALE", que es a la que se desea transferir el control cuando la opción proporcionada sea inválida. Por lo tanto, el proceso final quedaría de la siguiente forma sin incluir las demás partes, las cuales deberán ser iguales a las mostradas al inicio del presente ejemplo.

```
ECHO OFF
IF "%1" == "" GOTO AYUDALE
SET ETIQUETA=AYUDALE
FOR %%O IN (INSERTA inserta MUESTRA muestra IMPRIME imprime
            BORRA borra) DO IF %1 == %%O SET ETIQUETA=%1
GOTO %ETIQUETA%
:AYUDALE
```

Este mecanismo de identificación es muy eficiente y tiene la ventaja adicional de que emplea el mismo tiempo en identificar cualquiera de las opciones de la lista. Si un programa BATCH en particular tuviera tantas opciones que la lista completa de ellas no cupiera en un sólo comando FOR deberán utilizarse dos comandos FOR juntos, cada uno con la mitad de las opciones posibles, pero tal situación sólo se presentará en muy raras ocasiones.

3.6 Tercer ejemplo: Simulación de subrutinas

En este tercer y último ejemplo de uso de variables se describe el mecanismo general de operación de las subrutinas y la forma de simularlo apoyándose en el uso de variables BATCH.

Durante el desarrollo de programas grandes con frecuencia se presenta la situación de que ciertas partes de un programa se repiten en dos o mas ocasiones en forma idéntica, o por lo menos muy parecida. Una conclusión obvia después de observar esta

situación sería el preguntarse si no habrá forma de aprovechar lo que ya se escribió una vez sin necesidad de repetirlo tantas veces como veces se vaya a utilizar.

El poder hacer lo anterior representaría varias ventajas, entre las que se encuentran las siguientes: la sección del programa encargada de realizar el proceso que se repite se escribiría una sola vez en lugar de muchas, lo que disminuiría el tamaño del programa, y por lo tanto, su tecleo y la posibilidad de cometer errores durante el mismo; además, como el programa resultante sería mas pequeño y en las partes antes mencionadas sólo habría una instrucción encargada de ejecutar la sección completa, el programa sería mas fácil de entender y modificar; finalmente, si hubiera necesidad de corregir la sección que se repite sólo sería necesarió modificar una parte del programa, en lugar de todas las partes en las que esa sección estuviera repetida.

Este problema es común a todos los lenguajes de programación y la forma de resolverlo es mediante el siguiente procedimiento: se identifica la sección que se repite en el programa y se extrae del mismo, colocándola en un sitio aparte denominado "subrutina"; posteriormente se eliminan todas las secciones repetidas del programa y en su lugar de deja una instrucción especial denominada "invocación" de la subrutina.

La invocación de la subrutina deberá ejecutar todas las instrucciones contenidas en ella de tal forma que cuando la subrutina termine se continúen ejecutando las instrucciones siguientes a su invocación, con lo cual todas las instrucciones se ejecutarán en el mismo orden que antes, cuando las secciones no se habían extraido y formaban parte del programa original.

Consideremos el siguiente programa BATCH, el cual tiene las tres secciones indicadas repetidas con las mismas instrucciones:

```
   ECHO OFF
   REM SE PIDE EL NUEVO DISCO PARA FORMATEARLO
   CLS
   ECHO Inserte el nuevo disco en la unidad B:
/  PAUSE
\  ECHO Espere un momento por favor
   FORMAT B:
   REM SE PIDE EL DISCO ORIGINAL
   CLS
   ECHO Inserte el disco de datos en la unidad B:
/  PAUSE
\  ECHO Espere un momento por favor
   COPY B:*.*
   REM SE PIDE OTRA VEZ EL NUEVO DISCO PARA COPIAR
   CLS
   ECHO Inserte el nuevo disco en la unidad B:
/  PAUSE
\  ECHO Espere un momento por favor
   COPY *.* B:
```

Si extraemos las secciones repetidas para definir una subrutina, ésta se compondría de las siguientes dos líneas:

```
PAUSE
ECHO Espere un momento por favor
```

Con lo cual, el programa original quedaría de la siguiente forma:

```
ECHO OFF
REM SE PIDE EL NUEVO DISCO PARA FORMATEARLO
CLS
ECHO Inserte el nuevo disco en la unidad B:
-> invocación de la subrutina
FORMAT B:
REM SE PIDE EL DISCO ORIGINAL
CLS
ECHO Inserte el disco de datos en la unidad B:
-> invocación de la subrutina
COPY B:*.*
REM SE PIDE OTRA VEZ EL NUEVO DISCO PARA COPIAR
CLS
ECHO Inserte el nuevo disco en la unidad B:
-> invocación de la subrutina
COPY *.* B:
```

Si analizamos las instrucciones que quedan en este programa nos damos cuenta que las dos líneas que preceden a cada invocación de la subrutina son casi iguales. En efecto, en los tres casos se trata de un comando CLS seguido por un comando ECHO, sin embargo, la diferencia está en el mensaje particular que éste muestra.

Si en varias secciones se repite un proceso similar, aunque con datos diferentes, la subrutina también podrá ejecutar ese proceso auxiliándose de un valor que le indique el dato particular que deberá procesar en cada ocasión; a ese dato se le llama "parámetro" de la subrutina y en este caso será posible simularlo mediante el uso de una variable BATCH. Por lo tanto, la subrutina anterior incluiría también a los comandos CLS y ECHO mencionados aunque este último deberá mostrar el valor de una variable auxiliar que hará las veces del parámetro de la subrutina, por lo que ésta quedaría con los siguientes cuatro comandos:

```
CLS
ECHO %PARAMETRO%
PAUSE
ECHO Espere un momento por favor
```

Mientras que el programa original se reduciría a lo siguiente:

```
ECHO OFF
REM SE PIDE EL NUEVO DISCO PARA FORMATEARLO
SET PARAMETRO=Inserte el nuevo disco en la unidad B:
-> invocación de la subrutina
```

```
    FORMAT B:
    REM SE PIDE EL DISCO ORIGINAL
    SET PARAMETRO=Inserte el disco de datos en la unidad B:
 -> invocación de la subrutina
    COPY B:*.*
    REM SE PIDE OTRA VEZ EL NUEVO DISCO PARA COPIAR
    SET PARAMETRO=Inserte el nuevo disco en la unidad B:
 -> invocación de la subrutina
    COPY *.* B:
```

Ahora bien, ¿cómo se logra que al terminar la ejecución de una subrutina el proceso regrese al punto siguiente a la invocación de la misma?. Antes que nada, es importante aclarar que el lenguaje BATCH cuenta con verdaderas subrutinas en las cuales el manejo de esta situación es automático, y que lo que estamos haciendo en este momento es simplemente describir su mecanismo general de operación y tratar de simularlo de alguna manera. Por supuesto, al utilizar subrutinas en BATCH no emplearemos este método sino el que se describe en el siguiente capítulo; sin embargo, esta simulación es conveniente porque facilita la comprensión del uso de las subrutinas verdaderas.

Para invocar una subrutina se podría trasferir el control hacia ella simplemente con un comando GOTO; sin embargo, cuando la subrutina termine deberá regresar a un punto que no será el mismo en sus diversas ejecuciones. El problema consiste entonces en que la subrutina debe "recordar" el punto al cual debe regresar, y que ese punto estará marcado por el sitio en el programa original desde el cual se invoque la subrutina.

Por lo tanto, para resolver finalmente este problema se podrían colocar diversas etiquetas en el programa principal de manera que marquen los puntos a los que debe regresar la subrutina en cada invocación particular. De esta manera bastará con que a una cierta variable BATCH, a la cual le podremos llamar "REGRESO", se le asigne la etiqueta a la cual debe regresar la subrutina en cada ocasión, y terminar la ejecución de la misma con un comando GOTO hacia el contenido de la variable antes indicada.

A continuación se muestra la versión final del programa incluyendo las últimas modificaciones mencionadas. Hay que notar que, puesto que las instrucciones que componen la subrutina están colocadas dentro del mismo programa ya que no hay otro lugar dónde ponerlas, será necesario saltárselas cuando el programa termine, ya que de lo contrario se ejecutarían nuevamente al final del mismo sin que hubiera de por medio ninguna invocación.

```
    ECHO OFF
    REM SE PIDE EL NUEVO DISCO PARA FORMATEARLO
    SET PARAMETRO=Inserte el nuevo disco en la unidad B:
    SET REGRESO=ETIQ-1
 -> GOTO SUBRUTIN
    :ETIQ-1
    FORMAT B:
```

```
   REM SE PIDE EL DISCO ORIGINAL
   SET PARAMETRO=Inserte el disco de datos en la unidad B:
   SET REGRESO=ETIQ-2
-> GOTO SUBRUTIN
   :ETIQ-2
   COPY B:*.*
   REM SE PIDE OTRA VEZ EL NUEVO DISCO PARA COPIAR
   SET PARAMETRO=Inserte el nuevo disco en la unidad B:
   SET REGRESO=ETIQ-3
-> GOTO SUBRUTIN
   :ETIQ-3
   COPY *.* B:
   REM FINALMENTE, SI BRINCAN LAS INSTRUCCIONES DE LA SUBRUTINA
   GOTO FIN

   REM EN ESTE LUGAR SE COLOCA LA SUBRUTINA
   :SUBRUTIN
   CLS
   ECHO %PARAMETRO%
   PAUSE
   ECHO Espere un momento por favor
   REM LA SUBRUTINA REGRESA AL PUNTO DE INVOCACION
   GOTO %REGRESO%

   REM FIN DEL PROGRAMA, SE BORRAN LAS VARIABLES UTILIZADAS
   :FIN
   SET PARAMETRO=
   SET REGRESO=
```

Aunque finalmente el programa quedó mas grande y probablemente mas complicado que el programa original no es eso lo importante en este momento, sino que se haya comprendido el mecanismo general del funcionamiento de las subrutinas. Es recomendable que el lector revise la ejecución del programa anterior línea por línea apuntando el valor de las variables y siguiendo el flujo del programa a lo largo de los diversos comandos GOTO.

Es conveniente mencionar de nuevo que el uso de las verdaderas subrutinas no implica tantas molestias y detalles como los que se han visto en este ejemplo, aunque su mecanismo de funcionamiento es por completo similar al aquí descrito. En el siguiente capítulo se describe la forma real de utilizar las verdaderas subrutinas en el lenguaje de programación BATCH.

Capítulo 4 - MANEJO DE SUBRUTINAS BATCH

4.1 Utilización de subrutinas

De acuerdo a lo mencionado en la sección anterior, una subrutina consiste en una pequeña sección de un programa que se aísla del programa original y cuya utilización tiene una serie de ventajas, como sería disminuír el tamaño del programa y facilitar su tecleo y la corrección de errores dentro de la subrutina.

Sin embargo, probablemente la mayor ventaja del uso de subrutinas en general sea el hecho de que por medio de ellas será posible subdividir un programa grande y complicado en una serie de secciones pequeñas, permitiendo comprenderlas de mejor manera así como también al programa original que se reducirá a una serie de invocaciones a subrutinas con nombres adecuados. Si se piensa desarrollar un programa grande y complicado, el tratar de tomar en consideración todos los detalles de su diseño al mismo tiempo puede ser una tarea extenuante; en cambio, si se divide el programa grande en pequeñas secciones, el desarrollar cada una de estas secciones en forma individual constituirá una labor mucho mas sencilla y con menos errores. Por lo tanto, el uso de subrutinas debe ser considerado como una técnica que permite el desarrollo de programas grandes en una forma sencilla y correcta.

Por otro lado, puesto que el lenguaje BATCH por sí mismo no cuenta con características muy poderosas, las subrutinas serán el único medio de poder realizar ciertos procesos avanzados, los cuales se describirán posteriormente con ejemplos específicos.

Ya entrando en materia, una subrutina BATCH se forma con un archivo de comandos BATCH en forma similar a cualquier otro programa BATCH, por lo que en este caso los comandos que forman la subrutina efectivamente se encuentran en un sitio aparte del programa original, el cual es un archivo BATCH diferente.

De acuerdo a lo indicado en la sección anterior, el mecanismo de funcionamiento de las subrutinas consiste en la subrutina propiamente dicha, la cual es un archivo BATCH, y una forma especial de utilizarla llamada "invocación" de la subrutina, la cual deberá activar la subrutina de tal forma que cuando ésta termine su ejecución se deberá proseguir con los comandos colocados enseguida de la invocación de la subrutina.

Puesto que la subrutina es un programa BATCH como cualquier otro, terminará cuando se ejecute el último de sus comandos; ahora bien, la invocación de la subrutina se hace por medio del comando COMMAND /C en la forma descrita a continuación.

4.2 Comando COMMAND

Forma de uso: COMMAND /C subrutina parámetros

Invoca una subrutina.

El comando COMMAND /C permite la invocación de una subrutina colocada en otro archivo BATCH a la cual le transferirá los parámetros colocados enseguida de su nombre mediante el mecanismo descrito con anterioridad en la sección 2.2.

El programa que se ejecuta originalmente se denomina "programa principal" el cual puede a su vez mandar ejecutar a alguna subrutina. Un programa principal se ejecuta en forma directa, es decir, insertando su nombre desde el teclado, mientras que una subrutina se ejecuta cuando la invoca un programa principal, o bien, cuando la invoca a su vez otra subrutina.

Observe y analice los dos siguientes archivos BATCH.

Programa principal de ejemplo: EJ-PROG1.BAT

```
ECHO OFF
CLS
ECHO EJEMPLO DE INVOCACION A SUBRUTINA
ECHO PROCEDO A INVOCAR A LA SUBRUTINA EJ-SUBR1.BAT
ECHO PASANDOLE EL PARAMETRO: "PRIMERO"
COMMAND /C EJ-SUBR1 PRIMERO
ECHO REGRESE DE LA SUBRUTINA EJ-SUBR1.BAT
ECHO PROCEDO A INVOCARLA NUEVAMENTE
ECHO PASANDOLE ESTA VEZ EL PARAMETRO: "SEGUNDO"
COMMAND /C EJ-SUBR1 SEGUNDO
ECHO REGRESE POR SEGUNDA OCASION DE LA SUBRUTINA
ECHO FIN DEL EJEMPLO
```

Subrutina de ejemplo: EJ-SUBR1.BAT

```
ECHO OFF
ECHO SOY LA SUBRUTINA %0
ECHO RECIBI EL PARAMETRO: "%1"
ECHO PROCEDO A REGRESAR AL PROGRAMA QUE ME INVOCO
```

Si en el programa principal la subrutina se hubiera activado simplemente con el comando EJ-SUBR1 PRIMERO, sin incluir el COMMAND /C, este archivo también se hubiera ejecutado, pero al terminar no se hubiera regresado al programa principal sino que la ejecución se hubiera terminado en ese punto, por lo que no se ejecutaría ninguno de los comandos colocados después de la activación de EJ-SUBR1 hasta el fin del programa principal.

Cuando un programa BATCH se activa desde otro de esta forma, es decir, colocando su nombre sin incluir el COMMAND /C, al segundo programa ejecutado se le denomina "overlay". Cuando un overlay termina su ejecución el proceso termina en ese punto, como si hubiera terminado el programa principal activado originalmente. (Desgraciadamente, en computación es necesario en ocasiones manejar términos en inglés, por lo que a pesar de que en este caso el autor sugiere el término "programa de relevo" en sustitución de "overlay", se manejarán términos en inglés en la descripción de este tema y en diversos temas posteriores.)

Por lo tanto, la diferencia que se obtiene al invocar a otro programa mediante COMMAND /C es el hecho de que al terminar la ejecución de ese programa se continuará con la ejecución de las líneas siguientes al COMMAND /C en el programa principal, en cuyo

caso al programa activado se le llama subrutina. Si el programa se ejecuta directamente, sin el COMMAND /C, cuando termine terminará todo el proceso, en cuyo caso se le llama overlay.

Ahora bien, las subrutinas se pueden aprovechar de diversas formas aunque no se tengan varias secciones repetidas en un mismo programa. Suponga que se desea hacer un programa BATCH que por cada archivo seleccionado por el comodín dado por el primer parámetro, lo copie al disco de la unidad B: y enseguida lo borre de la unidad actual. Aparentemente esto es sencillo, veamos:

```
FOR %%F IN (%1) DO COPY %%F B:  y además  DEL %%F
```

Lo anterior no es posible hacerlo de esta forma, ya que el comando FOR puede ejecutar a su vez un sólo comando y en este caso se desean ejecutar dos. Una primer solución podría ser el incluir ambos comandos en otro archivo BATCH, llamado por ejemplo MUEVE.BAT, y mandarlo ejecutar desde el FOR de esta manera:

```
FOR %%F IN (%1) DO MUEVE %%F
```

Sin embargo, esto sólo funcionaría con el primer archivo seleccionado por el comodín, ya que al ejecutar al programa MUEVE.BAT en forma de overlay al terminar de copiar y borrar al primer archivo el proceso terminaría en ese punto, sin regresar al comando FOR que lo activó desde el programa principal.

Lo que se necesita es que una vez que el programa MUEVE.BAT termine su labor se regrese al programa principal a tomar el siguiente archivo proporcionado por el comando FOR. Puesto que ese es precisamente el funcionamiento de las subrutinas, para resolver este problema basta con invocar a MUEVE.BAT en forma de subrutina mediante COMMAND /C:

```
FOR %%F IN (%1) DO COMMAND /C MUEVE %%F
```

Esta situación también se puede aplicar al comando IF, es decir, si en un comando IF se desean ejecutar varios comandos cuando la condición que establece el IF se cumpla, inclúyanse todos los comandos deseados en otro archivo BATCH y ejecútese éste desde el IF invocándolo en forma de subrutina mediante COMMAND /C.

Un beneficio adicional que se tiene al utilizar subrutinas es que, puesto que la subrutina es otro programa que se ejecuta en forma separada del programa principal, si se cancela la ejecución de la subrutina no se cancela el proceso global, sino que simplemente se pasa a la instrucción siguiente a la invocación de la subrutina como si ésta hubiera terminado normalmente. Esta característica puede ser aprovechada en la ejecución selectiva de procesos sobre una serie de archivos, permitiendo o cancelando un cierto proceso sobre cada archivo en una subrutina, y procesando de todas formas todos los archivos indicados por algún comodín.

El siguiente programa de ejemplo, junto con su subrutina acompañante, borra en forma selectiva los archivos indicados por el comodín proporcionado por el primer parámetro, permitiendo que el usuario elija los archivos que serán borrados y los que no.

Programa de ejemplo: EJ-PROG2.BAT

```
ECHO OFF
CLS
REM POR CADA ARCHIVO SE EJECUTA LA SUBRUTINA EJ-SUBR2.BAT
FOR %%F IN (%1) DO COMMAND /C EJ-SUBR2 %%F
```

Subrutina de ejemplo: EJ-SUBR2.BAT

```
ECHO OFF
ECHO ¿DESEA BORRAR AL ARCHIVO %1?
ECHO ( CTRL-C = NO, CUALQUIER OTRA TECLA = SI )
PAUSE
DEL %1
```

Esta última característica no se presenta en la ejecución de un overlay, ya que si un programa principal ejecuta a un overlay y éste se cancela, se terminará la ejecución de todo el proceso como si el programa cancelado hubiera sido el programa principal.

Hay que notar que cuando se invoca una subrutina ésta comienza su ejecución mostrando los comandos que ejecuta aunque en el programa principal se haya apagado el eco mediante un ECHO OFF, es decir, la opción de eco es independiente para cada subrutina ejecutada, lo cual obliga a incluir un comando ECHO OFF a su vez al inicio de cada subrutina; sin embargo, el propio comando ECHO OFF sí aparecerá en la pantalla, lo cual pudiera ser un problema en ciertas aplicaciones BATCH avanzadas.

4.3 Comando PROMPT

Forma de uso: PROMPT=texto-indicador

Permite rastrear la ejecución del programa.

Cuando se desarrolla un programa grande en el lenguaje BATCH es conveniente probar por completo todas sus partes para confirmar que funcionen correctamente. Si el programa tiene una forma complicada de operar o está compuesto por una gran cantidad de subrutinas, sería conveniente saber en todo momento cual es la sección del programa que está en ejecución; a esta facilidad se le conoce como "rastreo" de la ejecución de un programa.

De acuerdo a lo indicado en la sección 1.4 referente al comando ECHO, si el eco está activo el sistema operativo MS-DOS mostrará cada comando en la pantalla antes de ejecutarlo precedido de un "texto indicador", que normalmente serán los caracteres "A>",

pero que podrá modificarse por medio del comando PROMPT, lo que permitirá cambiar el texto indicador de acuerdo a cada una de las partes del programa que se estén ejecutando en un momento dado.

Por ejemplo, considere la siguiente parte de un programa:

```
REM REVISA SI EXISTE EL ARCHIVO DE DATOS
IF EXIST DATOS.DAT GOTO SIGUE
PROMPT=NO HAY DATOS$G
REM PIDE EL DISCO DE DATOS PARA COPIAR EL ARCHIVO
ECHO INSERTE EL DISCO "DATOS" EN LA UNIDAD B:
PAUSE
COPY B:DATOS.DAT
ECHO YA PUEDE GUARDAR EL DISCO DE DATOS
PROMPT=
:SIGUE
```

Si el archivo DATOS.DAT existe en el disco se pasará a la línea SIGUE, de lo contrario se ejecutará la sección colocada en seguida del comando IF, en cuyo caso se cambiará el texto indicador que será mostrado antes de los comandos colocados en esta sección por los caracteres "NO HAY DATOS>". Hay que notar que dentro de los caracteres que definen el texto indicador no se podrá colocar directamente el caracter ">", por lo que para mostrarlo será necesario incluir en ellos el par de caracteres "$G". Al terminar esa sección el texto indicador se regresará a su forma usual con el comando PROMPT=.

Si el programa está formado por múltiples subrutinas que se invocan una a la otra será conveniente que el texto indicador muestre el nombre de la subrutina que se encuentre en ejecución en todo momento, lo cual puede hacerse incluyendo el siguiente comando al principio de cada subrutina:

```
PROMPT=%0$G
```

Será conveniente incluir el comando anterior al principio del programa principal y de toda subrutina durante la fase de pruebas y desarrollo, y cambiarlo posteriormente por el comando ECHO OFF una vez que el sistema esté terminado y listo para ser utilizado.

El manual del sistema operativo MS-DOS describe otros caracteres especiales que se pueden utilizar en la secuencia de caracteres que define el texto indicador, además de $G, que servirán para mostrar otra información importante con el comando PROMPT.

4.4 Comando EXIT

Forma de uso: EXIT

Termina la ejecución de una subrutina.

Es frecuente que en las subrutinas BATCH se incluya algún comando IF destinado a detectar alguna condición que indique no proceder con el resto de la subrutina, y que generalmente ejecuta un comando GOTO que transfiere la ejecución hacia una línea colocada al final de la subrutina BATCH. Por ejemplo:

```
IF "%1" == "" GOTO FIN
```

Sin embargo, si la subrutina tiene una gran cantidad de líneas la ejecución del GOTO anterior puede ser muy tardada, ya que MS-DOS debe leer el archivo de comandos BATCH de la subrutina desde el principio y línea por línea en busca de la etiqueta :FIN, y si la etiqueta está colocada al final del archivo el tiempo invertido en llegar a ella puede ser considerable, sobre todo si el archivo de la subrutina se encuentra en discos flexibles.

El comando EXIT proporciona una alternativa rápida para terminar la ejecución de una subrutina BATCH. Este comando puede ser ejecutado en cualquier lugar dentro de una subrutina, p.e.:

```
IF "%1" == "" EXIT
```

El comando EXIT no tendrá ningún efecto si se ejecuta desde un programa principal.

4.5 Comando CALL

Forma de uso: CALL subrutina parámetros

Invoca una subrutina interna.

Antes que nada, es importante remarcar que el comando CALL se incluyó hasta la versión 3.3 del sistema operativo MS-DOS, por lo que la presente descripción no se aplica si la computadora no cuenta con esta versión de MS-DOS o alguna versión posterior.

El objetivo del comando CALL es por completo similar al del comando COMMAND /C en el sentido de que lo que se busca es ejecutar las instrucciones contenidas dentro de un programa aparte, llamado subrutina, de tal forma que al terminar éste se continúen ejecutando las instrucciones colocadas después de la invocación de la subrutina en el programa principal.

Sin embargo, existen importantes diferencias de operación en la ejecución de ambos comandos derivadas del hecho de que el comando CALL es un comando interno de MS-DOS, mientras que COMMAND /C implica la ejecución del archivo COMMAND.COM en forma similar a cualquier otro comando externo. A continuación se mencionan las diferencias existentes entre COMMAND /C y CALL.

El comando CALL fué incluído con el fin de agilizar la ejecución de las subrutinas, ya que a pesar de que el mecanismo funciona perfectamente mediante COMMAND /C, la ejecución de éste implica

leer de disco un archivo que mide entre 20 y 30 K-bytes, lo cual es un proceso tardado. Debido a esto, la sustitución de un comando COMMAND /C que se utilice frecuentemente por un CALL puede implicar un importante ahorro en el tiempo involucrado en la ejecución de un determinado programa BATCH. Por ejemplo, si un comando FOR invoca repetidas veces a una subrutina, la diferencia en tiempo al invocarla mediante COMMAND /C o CALL es muy notoria.

Sin embargo, puesto que el comando CALL es un comando interno a MS-DOS, el cual será el responsable de la ejecución tanto del programa principal como de la subrutina, la ejecución de una subrutina mediante CALL no la aísla del todo de ciertos detalles en la forma en que lo hace COMMAND /C, por lo que la subrutina comparte una serie de caracteristicas de operación con el programa que la invocó en forma similar a como lo hace un overlay. Debido a esto, una subrutina invocada mediante CALL se comportará como un overlay en lo referente a la cancelación de la ejecución de la subrutina y al acceso a las variables BATCH.

Por ejemplo, si una subrutina invocada mediante CALL se descontinúa con la tecla Ctrl-C, el control no regresará al programa que la invocó, por lo que en ese punto se terminaría la ejecución tanto de la subrutina como del programa principal. De la misma manera, si se ejecuta el comando EXIT dentro de una subrutina interna no tendrá ningún efecto, en forma similar a su ejecución desde un programa principal.

La diferencia mas importante entre COMMAND /C y CALL cuando se desarrollan programas BATCH grandes y avanzados es el hecho de que una subrutina invocada mediante CALL tiene acceso exactamente a las mismas variables BATCH que el programa principal, mientras que una subrutina invocada con COMMAND /C genera un duplicado de las variables del programa principal para ser utilizadas en forma privada o local por la nueva subrutina invocada.

Es por esta razón que en este manual se aplica el término "subrutina interna" a una subrutina invocada con CALL, ya que a pesar de estar en otro archivo BATCH su ejecución se lleva a cabo dentro del mismo contexto del programa principal, es decir, compartiendo las mismas variables y la misma forma de terminación que éste, mientras que una subrutina invocada mediante COMMAND /C está completamente separada del programa principal en este sentido, siendo por lo tanto una subrutina "externa" o verdadera.