di Francesco Petroni

Tecniche di Programmazione

Quinta parte

Come già detto più volte nel corso delle puntate precedenti il DB III permette un uso in comandi diretti con il quale, data la potenza e la versatilità degli stessi, si ottengono facilmente risultati anche complessi.

L'utilizzazione del DB come linguaggio di programmazione, necessaria per ambiti applicativi avanzati, consiste in pratica nella possibilità di raccogliere un insieme di comandi, utilizzabili anche direttamente, in un file eseguibile tramite lo specifico comando DO < nome del file> .

Esistono inoltre alcuni comandi che hanno senso solo se utilizzati in un programma.

Sono molto pochi, ma proprio per questo possono essere inseriti in svariati ambiti applicativi e quindi al di là della loro sintassi va capita principalmente la loro filosofia.

Obiettivo di questa puntata è appunto quello di esemplificare l'uso di tali comandi, i vari ambiti applicativi e di sottolineare la loro versatilità.

Attività preliminari

Il DB III è un Data Base Management System, ovvero un prodotto specializzato nella gestione di archivi. Qualsiasi attività si svolga con il DB III consiste nella manipolazione di dati presenti in uno o più archivi.

È quindi buona norma pratica, ogni volta che si sperimentano comandi o che si realizzano programmi, disporre di un archivio di test che contenga una tipologia di campi completa e che contenga un certo numero di record con i quali si riesca a simulare le varie problematiche applicative.

In pratica si crea un archivio, lo si carica con una ventina di record, poi se ne stampa la struttura e il contenuto e tali output si tengono ben in vista sul tavolo di lavoro, sia per poter scrivere in maniera sintatticamente corretta i vari comandi, sia per poter controllarne l'esattezza logica confrontando il risultato con i dati effettivamente in archivio.

In figure 2 e 3 visualizziamo per comodità struttura e contenuto del file che usiamo per gli esempi presentati in questa puntata.

Con l'occasione ricordiamo che il DB III è un interprete e quindi, essendo possibile eseguire direttamente i comandi, è possibile testarli anche prima di inserirli in un programma.

Si supponga di dover arrotondare una somma in lire alle 1000 lire inferiori o superiori a seconda che le ultime tre cifre siano o meno inferiori a 499 lire. La conseguente espressione matematica o funzione può essere testata in comandi diretti prima di essere inserita nel programma dove il test è più gravoso.

In figura 1 c'è l'hardcopy della videata del test eseguito in comandi diretti sia per testare tale espressione che per testare un insieme di funzioni di traduzione di una data. Se le espressioni «funzionano» in comandi diretti funzioneranno anche in un programma.

```
Struttura del database: A:archivio.dbf
Totale record inseriti:        24
Ultima revisione       : 01/01/80
Campo  Nome campo  Tipo campo    Dim     Dec
    1  NOME        Carattere      12
    2  COGNOME     Carattere      14
    3  SEX         Carattere       1
    4  DATANASC    Data            8
    5  PROVNASC    Carattere       2
    6  STIPENDIO   Numerico        8
Totale:                          46
```

Figura 2 - Struttura dell'Archivio Usato negli Esempi. Qualsiasi lavoro si voglia fare su un archivio bisogna averne sempre ben nota la struttura.

Record	NOME	COGNOME	SEX	DATANASC	PROVNASC	STIPENDIO
10	CHIARA	ROSA	F	25/03/60	MI	1456000
11	ALDO	AZZURRI	M	15/02/56	MI	2009000
12	DORINA	MALTESI	F	11/11/65	MI	4560000
13	LUISA	ROSSINI	F	19/06/55	RM	5200000
14	MARIO	LUSSU	M	22/03/65	PA	540900
15	LIDIO	BIANCHINI	M	29/05/51	LI	2190000
16	CARLOTTA	GIALLI	F	21/05/45	LI	1875000
17	MARGHERITA	VIOLA	F	03/10/54	RM	1090000
18	WALTER	LUCIOLI	M	11/02/55	RM	4100000
19	BIAGIO	MALETTI	M	19/02/57	RM	1295000

Figura 3 - Listato Parziale dell'Archivio di Prova. In sede di test sia usando i comandi diretti, che sotto programma occorre conoscere il contenuto dell'archivio di prova per verificare l'esattezza dei comandi impostati.

Il Comando DO WHILE .. ENDDO

Il primo comando specifico, utilizzabile solo in programmazione, è il DO WHILE .. ENDDO, che è ben noto anche a chi programma in Basic, nella forma WHILE .. WEND. La sua sintassi:

```
DO WHILE <condizione>
  ...
  insieme di comandi
  ...
ENDDO
```

Il significato è «esegui l'insieme dei comandi interni alla struttura finché non è più vera la condizione iniziale». Quando non è più vera la condizione iniziale il programma prosegue in sequenza.

Per poter entrare nella struttura DO WHILE .. ENDDO è necessario che almeno una volta, all'inizio, la condizione di entrata sia verificata. Per poterne uscire, oltre a sistemi diretti, che vedremo dopo, è necessario che «succeda qualcosa» che modifichi gli elementi della condizione sino a fare in modo che non si verifichi più.

I sistemi diretti, invece, sono costituiti da comandi che, indipendentemente dal verificarsi della condizione, provocano l'uscita dalla struttura, e sono due:

```
DO WHILE <condizione>
  ...
  RETURN
  ...
ENDDO
```

Se all'interno della struttura si incontra un RETURN, in genere legato anch'esso al verificarsi di una condizione il programma termina e se è stato richiamato da un programma a livello superiore vi ritorna.

```
DO WHILE <condizione>
  ...
  EXIT
  ...
ENDDO
```

Anche in questo caso si esce dalla struttura indipendentemente dal raggiungimento della condizione, il programma prosegue in sequenza eseguendo le istruzioni successive all'ENDDO.

Nell'ambito di tale struttura ne esiste un tipo molto particolare che viene spesso usata per creare un ciclo «eterno» dal quale si esce solo tramite RETURN o EXIT.

```
DO WHILE .T.
  ..
  insieme di comandi
  ..
ENDDO
```

La condizione iniziale è sempre e comunque verificata, si esce solo con modalità diretta.

Non esiste nessuna istruzione di salto né diretto né condizionato, per cui l'esperto in Basic si trova in difficoltà in quanto non può utilizzare il GOTO. Occorre creare delle strutture di DO WHILE .. ENDDO, anche a più livelli, con le quali il raggiungimento di subroutine si ottiene evitando di entrare nelle altre.

Non esiste neanche l'istruzione Basic

```
FOR I = I1 TO 12 STEP 13
  ...
NEXT I
```

Ma questa è facilmente simulabile con una struttura DO WHILE

```
I=I1
DO WHILE I<12
  ...
I=I1+13
ENDDO
```

Va subito detto per chi è abituato al Basic, che è un errore pensare in «Basic» e tradurre la soluzione in DB III. Bisogna arrivare a pensare direttamente in DB III e alla fine non si rimpiangeranno i GOTO e i FOR .. NEXT.

Nel listato del programma che produce un Loop e nell'output relativo di figura 4 si può anche verificare l'uso del comando ?, del tutto equivalente al Basic PRINT, comando di visualizzazione semplice, che accetta variabili e virgole per separatori.

Il formato della variabile è quello di default, ma può essere corretto per mezzo di funzioni come la STR (X,4) che traduce in una stringa la variabile numerica X.

Il comando di print ? non accetta però la specifica PICTURE che permette di formattare stringhe, numeri e date nei più svariati modi. Questa specifica è utilizzabile solo con il comando evoluto di visualizzazione @ V,O SAY.

Il comando ? accetta le virgole come separatori, ma non accetta che sia-

```
. x=1234499
1234499
. ? round(x/1000,0)*1000
        1234000.00
. x=1234501
1234501
. ? round(x/1000,0)*1000
        1235000.00
. a=ctod("21/02/86")
21/02/86
. ? cdow(a)
Venerdi
. ? day(a)
 21
. ? cmonth(a)
Febbraio
. ? year(a)
 1986
. ? cdow(a)+str(day(a),3)+";
  "+cmonth(a)+str(year(a),6)
Venerdi 21 Febbraio  1986
.
```

Figura 1 - Test di alcune espressioni e funzioni. Prima di inserire espressioni e/o funzioni in un programma è opportuno eseguirne un test in comandi diretti. Se funzionano in comandi funzioneranno anche nel programma.

```
* 21/02/86 p5_0
* simulazione FOR  STEP  NEXT
SET TALK OFF
CLEAR
I1=2
I2=36
I3=3
I=I1
DO WHILE I<I2
 P=I*2
 Q=I**2
 D=I/2
 ? I,P,Q,D
 I=I+I3
ENDDO
 2          4          4.00          1.00
 5         10         25.00          2.50
 8         16         64.00          4.00
11         22        121.00          5.50
14         28        196.00          7.00
17         34        289.00          8.50
20         40        400.00         10.00
23         46        529.00         11.50
26         52        676.00         13.00
29         58        841.00         14.50
32         64       1024.00         16.00
35         70       1225.00         17.50
```

Figura 4 - Listato e output del Programma P5_0 Creazione di Un LOOP. Il programma rappresenta la traduzione del comando FOR .. NEXT del Basic in un programma DB III che utilizza la struttura DO WHILE .. ENDDO, che ha un significato molto più generale e quindi adattabile a svariati ambiti applicativi.

```
* 22/02/86 p5.1
* corso DBIII MC Micro Computer
*
clear
set talk off
l=chr(196)
c=0
* loop contatore
do while c<77
 l=l+chr(196)
 c=c+1
enddo
*
use archivio
go top
c=7
@ 1,1  say l
@ 2,1  say "M.C. Micro Computer - Corso DB III"
@ 2,40 say "Programma Scorrimento Archivio"
@ 4,1  say "Num.Rec. Nominativo"
@ 4,46 say "Giorno Mese e Anno di Nascita"
@ 5,1  say l
@ 22,1 say l
* loop scorrimento
do while .not. eof()
  if c=20 .or. eof()
    @ 22,1 say l
    r=" "
    * loop controllo
    do while .not. r$"PF"
      @ 23,10 say "Premi   P-prosegui   F-finisci  " get r pict "!"
      read
    enddo
    if r="F"
      clear
      return
    endif
    @ 6,0 clea
    c=7
  endif
  @ c,1  say str(recno(),5)
  @ c,10 say cognome
  @ c,20 say nome
  @ c,46 say cdow(datanasc)+" "+str(day(datanasc),2)
  @ c,62 say cmonth(datanasc)
  @ c,72 say year(datanasc)
  c=c+1
skip
enddo
@ 22,1 say l
wait
use
clear
```

Figura 5 - Listato del Programma P5_1 Scorrimento degli archivi. Il programma presenta ben tre ambiti applicativi del comando di programmazione DO WHILE <condizione> ENDDO, insieme al «collega» IF .. ELSE .. ENDIF comandi fondamentali di programmazione.

no inserite nella stessa istruzione differenti tipi di variabile, come ad esempio una stringa un numero e una data. La soluzione è quella di ridurre numero e data a stringa con le specifiche funzioni che adempiono a questo compito.

La specifica di formato PICTURE, che come detto offre numerosissime varianti, non è utilizzabile neppure con i comandi LIST, DISPLAY né e questo è più grave, con la struttura REPORT, rendendola pressoché inutile per stampe in cui riportare valori numerici con molte cifre.

In figura 5 presentiamo un programma, il cui output è in figura 6, in cui sono raccolti tre possibili differenti utilizzazioni del comando DO WHILE .. ENDDO.

Il primo uso è nella realizzazione di un contatore. Viene definita la variabile C, che viene incrementata all'interno della struttura, dalla quale si esce solo quando il contatore supera un determinato valore. Il contatore nel nostro caso viene utilizzato per definire una variabile L alla quale viene assegnata una stringa di 76 trattini orizzontali (carattere 196). Il secondo uso è legato al raggiungimento della condizione logica di EOF(), che serve per terminare lo scorrimento dei record di un file. Per generare lo scorrimento si usa la funzione skip, che sposta di un record il puntatore sul file.

Va notato come la condizione EOF() che altri non è che una funzione di sistema del DB III, è raggiunta non quando il puntatore è sull'ultimo record, ma quando si cerca di superarlo. Questo permette di gestire lo scorrimento anche dell'ultimo record, che altrimenti ne rimarrebbe escluso.

La terza modalità di uso del DO WHILE .. ENDDO è legata ad una problematica di controllo di un campo in input. In pratica si vuol controllare che alla variabile R sia assegnato il valore P o F.

L'istruzione di input (GET R) viene inserita all'interno di un ciclo DO WHILE .. ENDDO in cui la condizione di uscita è che la variabile R sia o P o F.

Viene utilizzata la funzione logica di ricerca di sottostringa $ che fornisce un valore .T. vero o .F. falso a seconda che la ricerca sia positiva o meno. La sua sintassi è:

```
? «come»$«ciao come stai»
```

Il risultato è .T. in quanto la stringa «come» è contenuta nella stringa «ciao come stai». Altra funzione di ricerca di sottostringa è la AT, che restituisce il valore numerico della posizione della prima stringa nella seconda. La sintassi:

```
? AT(«come»,«ciao come stai»)
```

Se la prima stringa non è contenuta nella seconda il valore restituito è zero, che ha il significato di non c'è.

Il programma P5_1 mostra altre cose interessanti come la tecnica per produrre lo scorrimento comandato da tastiera del file, e gestito tramite un contatore di riga C, che se raggiunge il valore 20 blocca lo scorrimento e chiede conferma per continuare.

Vengono inoltre utilizzate buona parte delle funzioni di data.

Il campo data non è né tipo stringa né tipo numerico. Le funzioni di data restituiscono o stringhe (nome dei giorni o dei mesi) o numeri (numero del giorno della settimana o del mese, oppure numero del mese o anno nella forma a quattro cifre).

Esistono anche delle funzioni inverse, per cui data una stringa di adeguato formato è possibile tradurla in una variabile o in un campo di tipo data.

Generatore di valori RANDOM

In DB III manca la funzione RANDOM con la quale vengono generati

```
M.C. Micro Computer - Corso DB III      Programma Scorrimento Archivio

Num.Rec. Nominativo                          Giorno Mese e Anno di Nascita

    1    ROSSI          LUIGI          Lunedi  9   Giugno     1958
    2    VERDI          MARCO          Venerdi 20  Marzo      1964
    3    BIANCHI        LUDOVICO       Martedi 29  Maggio     1951
    4    GIALLI         CARLOTTA       Lunedi  21  Maggio     1945
    5    VIOLA          MARGHERITA     Domenica  3 Ottobre    1954
    6    NERI           WALTER         Martedi 11  Gennaio    1955
    7    MARRONI        BENIAMINO      Sabato  29  Giugno     1957
    8    ARANCIO        GIUSEPPE       Giovedi  4  Settembre  1958
    9    BRUNO          MARIO          Giovedi 20  Dicembre   1962
   10    ROSA           CHIARA         Venerdi 25  Marzo      1960
   11    AZZURRI        ALDO           Mercoledi 15 Febbraio  1956
   12    MALTESI        DORINA         Giovedi 11  Novembre   1965
   13    ROSSINI        LUISA          Domenica 19 Giugno     1955


         Premi   P-prosegui   F-finisci
```

Figura 6 - Output su Video Ottenuto dal Programma P5_1. Lavorando in comandi diretti si può ottenere una specie di scroll su video inserendo particolari opzioni nei vari comandi. Confezionando un programma si ottiene un risultato più elegante controllando l'avanzamento tramite opportune istruzioni.

numeri casuali. Tale funzione è molto utile quando occorre realizzare simulazioni oppure quando, durante la fase di test dei programmi, occorre avere molti dati ma risulta gravoso «inventarseli».

Per mezzo della funzione RANDOM, usata insieme ad altre funzioni, si riesce a generare stringhe di caratteri, numeri di varia grandezza e formato, date, ecc., per cui può essere impiegata per caricare in maniera automatica archivi sui quali eseguire delle prove che simulino anche le più gravose condizioni di lavoro dei programmi e dei file.

Ad esempio se in un'applicazione è previsto che il file di lavoro contenga 10.000 record, il test deve essere eseguito con un file di prova di uguale importanza. Con un archivio di 10 record si testa la correttezza dei programmi ma non la effettiva prestazione della procedura.

In particolare via via che gli archivi si riempiono diminuiscono le prestazioni delle routine di ricerca, di indicizzazione, di selezione, e questo deve essere correttamente previsto e valutato.

In figure 7 e 8 presentiamo un programma generatore di numeri pseudo casuali ottenuti a partire da un numero iniziale (seme) e compresi tra 0 e 1. La letteratura tecnica presenta numerosi algoritmi per garantire il più possibile la casualità, ma nel nostro caso, non ci interessa più di tanto in quanto lo vogliamo usare non in procedure di simulazione, in cui è necessaria una reale casualità, ma per automatizzare il caricamento di un archivio.

```
* 22/02/86 p5_3
* generatore parole pseudo random
set talk off
clear
set decimal to 7
x=0
ni=0
@ 3,1 say "Immetti il Seme ( dieci valori) " get x pict "9999999"
@ 4,1 say "Numero delle Iterazioni          " get ni pict "999"
@ 6,1 say "-------------------------------"
@ 8,1 say "-------------------------------"
read
* apre archivio
use ar index in
i=1
do while i<ni+1
i=0
pa=""
nu=0
da=ctod("01/01/30")
* parola
do while i<10
 y=log(x)
 x=y-int(y)
 c=int(x*26+65)
 x=x*1000000
 pa=pa+chr(c)
 i=i+1
enddo
 @ 7,1 say str(i,5)
 @ 7,10 say pa
* numero
 y=log(x)
 x=y-int(y)
 nu=int(x*1000)*1000+1000000
 x=x*1000000
 @ 7,25 say nu pict "#,###,###,###"
* data
 y=log(x)
 x=y-int(y)
 da=da+int(x*10000)
 x=x*1000000
 @ 7,40 say da
*
i=i+1
append blank
replace codice with pa,numero with nu,data with da
enddo
@ 9,0
list
```

Figura 9 - Listato del Programma P5_3 Generatore Automatico Di Archivi. Il procedimento di randomizzazione viene usato per generare automaticamente delle stringhe che vengono caricate in un archivio indicizzato, e quindi ordinato sulla stringa stessa.

Figura 7 - Listato del Programma P5_2 Generatore Numeri Random. In DB3 si lamenta l'assenza di una funzione RANDOM che generi numeri casuali, funzione indispensabile per realizzare simulazioni e anche per caricare in modo automatico archivi casuali.

```
* 22/02/86 p5_2
* generatore numeri pseudo random
set talk off
clear
set decimal to 7
x=0000000
n=100
i=1
t=0
@ 3,1 say "Immetti il Seme ( dieci valori) " get x pict "########"
@ 4,1 say "Numero delle Iterazioni          " get n pict "###"
@ 6,1 say "-------------------------------"
@ 8,1 say "-------------------------------"
read
do whil i<n+1
 y=log(x)
 x=y-int(y)
 t=t+x
 @ 7,1  say str(i,5)
 @ 7,8  say x    pict "#.########"
 @ 7,24 say t/i pict "#.########"
 x=x*1000000
 i=i+1
enddo
```

```
Immetti il Seme ( dieci valori)  3434234
Numero delle Iterazioni          100

-------------------------------
   12  0.0030860       0.3938896
-------------------------------
```

Figura 8 - Output del Programma P5_2. Per verificare l'efficacia del metodo il programma genera un numero a richiesta di valori e ne fa la media, che effettivamente tende a .5.

In pratica dato il seme (un numero di 10 caratteri) se ne esegue il logaritmo la cui parte decimale è il numero casuale e poi viene riutilizzata come seme successivo. Quindi la casualità consiste solo nel fatto che nessuno si metterebbe a calcolare via via le parti decimali dei logaritmi.

Per testare la casualità estraiamo più numeri e ne facciamo la media, se questa tende a .5000 possiamo dichiararci soddisfatti.

Applichiamo immediatamente l'altogirtmo trovato per caricare automaticamente tre campi di un archivio. Il primo di tipo carattere lungo 10 caratteri, il secondo numerico il cui valore deve essere compreso tra 1,000,000 e 2,000,000 e il terzo di tipo data, con la caratteristica di essere compresa tra l'1/01/30 e la data corrispondente a 10,000 giorni dopo.

Il programma relativo, listato in figura 9, oltre alla parte algoritmica che utilizza varie funzioni di stringa e numeriche, prevede l'alimentazione di un archivio (si chiama «ar») già creato che comprenda i campi Codice, Numero e Data. Viene utilizzato anche un indice (si chiama «in») la cui chiave corrisponde al codice, per cui oltre all'archivio dati via via che vengono aggiunti record viene aggiornato anche l'indice.

Il caricamento avviene per mezzo delle due funzioni:

APPEND BLANK

che crea un nuovo record vuoto

REPLACE <campo> WITH <variabile>

con il quale le variabili comunque definite vengono trasferite nei corrispondenti campi dell'archivio.

Ripetiamo che l'obbiettivo di tale programma è quello di caricare in modo del tutto automatico un archivio. Il numero dei record immessi va stabilito via input (funzione GET della variabile NI).

Il vantaggio di tale sistema sta nel fatto che i record sono in genere differenti tra di loro (è improbabile generare stringhe e numeri uguali) e questo può simulare bene la reale situazione di lavoro.

Lo svantaggio invece consiste nel fatto che il caricamento è lento e quindi se si vogliono migliaia di record occorre lasciar lavorare il programma per ore.

In figura 10 viene presentato un metodo ultrarapido per realizzare archivi voluminosi partendo da un archivio piccolo e moltiplicandolo più volte. Lo svantaggio di questo secondo metodo sta nel fatto che i record saranno simili gli uni agli altri.

Il metodo consiste nell'aprire il file che ci interessa, nel nostro caso con 12 record, copiarlo in un altro file. Ricaricare dal secondo file sul primo tutti i record. Ricopiare di nuovo tutti i file sul secondo file, e così via.

Come si vede bene da tutti i messaggi che il DB III invia, il file originario raddoppia ad ogni passaggio la sua dimensione per cui in pochi passaggi si raggiungono volumi di migliaia di record. Il tempo di esecuzione rallenta via via in quanto variano le dimensioni. Se, come in questo caso non si utilizza un file indice, per caricare 1,000 record il tempo necessario è dell'ordine del minuto.

Va notato che l'esecuzione del programma va seguita perché quando si riversa il contenuto del primo file sul secondo il DB chiede una conferma. Va anche notato che il tutto avviene rimanendo sul primo file, il secondo viene smaneggiato senza aprirlo, al punto che finita l'operazione si può tranquillamente distruggerlo.

Per rimanere nell'argomento presta-

zioni non è possibile stabilire delle regole precise in quanto le variabili in gioco sono tante:

unità a disco in uso (hard/floppy)
grado di riempimento del supporto
prestazione dell'hardware (CPU e Drive)
memoria a disposizione
dimensioni della struttura
numero di indici attivi
tipo di algoritmo di caricamento

Alcune delle variabili in gioco sono facilmente modificabili con poca spesa, specie se la procedura che si sta realizzando deve diventare produttiva in un reale ambiente di lavoro. Ad esempio se il problema è la dimensione dell'archivio che è elevata per l'unità a disco di cui si dispone, la soluzione più economica è rappresentata dall'acquisto di un hard disk di adeguata capacità. Così come se il problema è la velocità di calcolo si risolve acquistando un compilatore per il linguaggio che si sta usando.

Comandi IF .. ELSE .. ENDIF
DO CASE .. ENDCASE

Oltre alla struttura DO WHILE .. ENDDO ne esistono altre due di grande importanza e che si usano pressoché in tutti i programmi. Anche queste sono ben note anche al più inesperto dei programmatori Basic e sono:

```
IF <condizione 1>
  serie di istruzioni 1
ELSE
  serie di istruzioni 2
ENDIF
```

```
* 22/02/86 p5_41
* esempio di if
do while .t.
v=" "
clear
do while .not. v$"ABCDF"
 @ 2,2  say "Risposte Possibili:
 A B C D e F (fine)   "
 @ 2,40 get v pict "!"
 read
enddo
if v="F"
 retu
endi
if v="A"
  @ 4,2 say "A"
else
  if v="B"
    @ 5,2 say "B"
  else
    if v="C"
      @ 6,2 say "C"
    else
      @ 7,2 say "D"
    endif
  endif
endif
wait
enddo
```

Figura 11 - Listato del Programma P5_41 IF .. ENDIF nidificati. Nidificando a vari livelli l'istruzione IF .. ENDIF si possono indirizzare anche insiemi complessi di condizioni. La difficoltà è come al solito quella di analizzare chiaramente tutti i casi e tutti i loro contrari.

```
* 22/02/86 p5_31
* caricamento automatico archivi
set talk on
clea
use arcuno
copy to arcdue
append from arcdue
copy to arcdue
append from arcdue
copy to arcdue
append from arcdue
copy to arcdue
append from arcdue
delete file arcdue.dbf
```

```
      12 record copiati.
      12 record aggiunti.
arcdue.dbf già esiste, va riscritto (S/N)? Si
      24 record copiati.
      24 record aggiunti.
arcdue.dbf già esiste, va riscritto (S/N)? Si
      48 record copiati.
      48 record aggiunti.
arcdue.dbf già esiste, va riscritto (S/N)? Si
      96 record copiati.
      96 record aggiunti.
Il file è stato eliminato.
```

Figura 10 - Listato e Out del Programma P5_31 Caricatore di Archivi. Se non interessa che esistano numerosi record uguali vi sono sistemi più economici per costruire archivi di prova di grosse dimensioni.

La serie di istruzioni 2 viene eseguito se non si verifica la condizione 1. È possibile costruire una struttura «semivuota» come:

```
IF <condizione 1>
ELSE
  serie di istruzione 2
ENDIF
```

Può essere utile quando è più facile esprimere una condizione tramite il suo contrario. È invece pericoloso esprimere la stessa struttura con:

```
IF <condizione 1>
  serie di istruzioni 1
ENDIF
IF <contrario condizione 1>
  serie di istruzioni 2
ENDIF
```

In quanto bisogna essere sicuri che i casi possibili siano tutti contemplati, e lo siano dove occorre e non altrove.

La struttura DO CASE .. ENDCASE rappresenta un IF più articolato in quanto può gestire più condizioni.

```
DO CASE
  CASE <cond. 1>
    istruzioni 1
  CASE <cond. 2>
    istruzioni 2
  CASE <cond. 3>
    istruzioni 3
ENDCASE
```

Le varie condizioni debbono escludersi mutuamente in quanto se si verifica la condizione 1, vengono eseguite le istruzioni 1 e, finite queste, il controllo passa direttamente all'ENDCASE. Ovvero se la condizione 2 è compresa nella 1, la serie di istruzioni 2 non sarà mai eseguita.

C'è una variante che permette di raggruppare le condizioni residue:

```
DO CASE
  CASE <cond. 1>
  ....
  ....
  OTHERWISE
    istruzioni n
ENDCASE
```

L'insieme delle istruzioni n viene eseguito solo se nessuna delle condizioni legate ai vari CASE è verificata.

L'applicazione più immediata del DO CASE è nei programmi di menu dove la variabile di input della selezione fatta in genere può assumere numerosi valori. In questo caso la struttura somiglia alla istruzione BASIC

ON X GOSUB i1,i2,i3,i4

In realtà la struttura può essere utilizzata in problematiche più complesse, insomma dove occorre districarsi tra numerose condizioni e sottocondizioni.

Il primo programma P5_41 (listato in figura 11), presenta una serie di IF nidificati a tre livelli, che permette di indirizzare 5 condizioni. Il secondo P5_42 (listato in figura 12) indirizza le stesse 5 condizioni del precedente, per mezzo di una struttura DO CASE, che ha un solo livello.

Appare evidente che la struttura DO CASE è più semplice e in definitiva permette una programmazione più pulita, senza pendenze in termini di ENDIF, ENDDO.

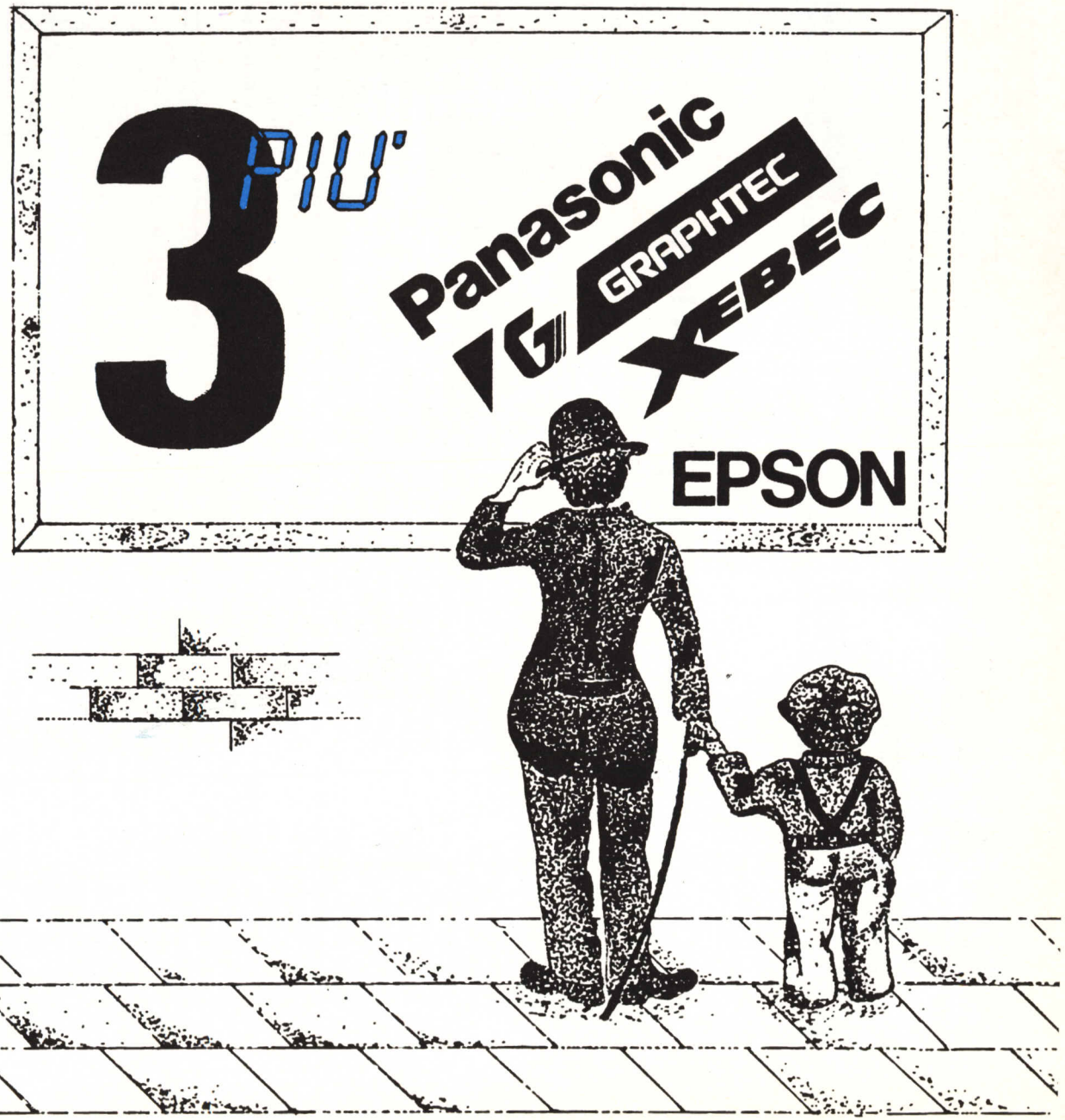
```
* 22/02/86 p5_42
* esempio di do case
do while .t.
v=" "
clear
do while .not. v$"ABCDF"
 @ 2,2  say "Risposte Possibili:
 A B C D e F (fine)   "
 @ 2,40 get v pict "!"
 read
enddo
DO CASE
 CASE v="F"
  retu
 CASE v="A"
  @ 4,2 say "A"
 CASE v="B"
  @ 5,2 say "B"
 CASE v="C"
  @ 6,2 say "C"
 CASE v="D"
  @ 7,2 say "D"
 ENDCASE
wait
enddo
```

Figura 12 - Listato del Programma P5_42 DO CASE .. ENDCASE. Una struttura più leggera per separare un insieme di casi individuabili ciascuno con una propria condizione, è fornita dalla DO CASE .. ENDCASE. Le varie condizioni sono mutuamente escludentesi.

.....finalmente sappiamo a chi rivolgerci per avere

COMPATIBILITÀ E QUALITÀ

- Sistemi Grafici Integrati
- Personal Computer
  IBM Compatibili & Portatili
- Stampanti
- Plotter & Digitizer
- Monitor Alta Risoluzione
- Terminali video
- Sottosistemi a Disco
  Winchester/Streamer
- Unità di Back-up
- Supporti Magnetici
  (5 1/4'' & 3 1/2'')



TREPIU'

INFORMATICA VIDEO TELEMATICA

TRE PIÙ S.r.l. - Telef. 06/5984841/2/3 (Ric. Aut.)
00144 Roma (Eur-Nir) Via Michelangelo Peroglio, 15