Five Solutions for the Three-level Control Break Problem

by Dr. Volker Thormählen

The purpose of this article is to present 5 equivalent solutions for the 3-level control break problem. Source code comes from *Visual Foxpro*, a powerful database and programming language from *Microsoft*. Adaptation of the solutions to handle a n-level (n = 1, 2, 3, …) control break is rather simple due to the regularity of programming techniques applied.

Sequential files are those that have been sorted into a specific sequence. Reports prepared from sequential files may require:

- Detail printing – containing information from each specific record.
- Group printing – containing information that summarizes specific records.

Consider the *Sales Report* shown in Figure 1. This report contains both, detail printing and group printing. Group totals are printed by branch, salesman, and customer:

- There may be one or more invoices for each customer. However, only one line should be print for each customer showing the total amount invoiced for that customer.
- Totals will be printed for each salesman and each branch. At the end of the report, the grand total of all sales will be printed.

Summary lines are marked by up to 3 asterisks depending on the level of the control break (minor, intermediate, or major).

```
                          SALES REPORT                          PAGE    1

BRANCH     SALESMAN    CUSTOMER    INVOICE        SALES AMOUNT
1          10          77650       50372              2.968,88
1          10          77650       56746                 55,46
1          10          77650       89805                485,83
1          10          77650                          3.510,17 *
etc.
3          18          77663                          4.066,85 *
3          18          77664       03911              2.344,15
3          18          77664       05456              2.809,69
3          18          77664       33888              3.390,48
3          18          77664                          8.544,32 *
3          18          77665       01399                368,14
3          18          77665       01695              5.590,97
3          18          77665                          5.959,11 *
3          18                                        84.888,20 **
3                                                   825.000,46 ***
                     GRAND TOTAL                  2.653.999,05
```

Figure 1: Illustrative cutting of *Sales Report* by branch, salesman, and customer

The structure of invoice records used for printing the Sales Report are shown in Figure 2. Corresponding records are sorted (indexed) in the following order:

1. Branch number - used for major break lines
2. Salesman number - used for intermediate break lines
3. Customer number - used for minor break lines
4. Invoice number - used for printing the invoice numbers in ascending order within a given customer number; not at all necessary for the 3-level control break logic.

Field	Field Name	Type	Width	Decimals
1	INVOICE	Character	5	
2	BRANCH	Character	1	
3	SALESMAN	Character	5	
4	CUSTOMER	Character	5	
5	ITEM	Character	5	
6	SALES	Numeric	13	2
Total			35	

Figure 2: Data structure of the invoice table

Five solutions for the three-level control break problem will be presented:

1. … using 4 nested DO loops (see Figure 3)
2. … using a modular program design (see Figure 4)
3. … using cascading procedures (see Figure 5)
4. … using 2 switches (see Figure 6)
5. . using a strict top-down approach (see Figure 7)

All five solutions produce equivalent output as illustrated in Figure 1.

Figure 8 contains lower-level routines such as NEW_PAGE. All of them are used commonly for the five control break programs.

A few documentary notes about each solution helps to explain the programming tasks involved. In all programs in this article, note how consistent and self-documenting the names are – those in procedures and data.

```
set talk off
clear
set procedure to cbreak
DO initialization
use invoice.dbf
index on branch+ salesman + customer + invoice tag combi_key
go top
DO while .not. eof()
   store 0 to total_3
   old_branch = branch
   DO while old_branch = branch .and. .not. eof()
      store 0 to total_2
      old_salesman = salesman
      DO while old_salesman = salesman .and.;
               old_branch   = branch .and. .not. eof()
         store 0 to total_1
         old_customer = customer
            DO while old_customer = customer .and.;
                     old_salesman = salesman .and.;
                     old_branch   = branch .and. .not. eof()
               DO process_detail
            ENDDO
            total_2 = total_2 + total_1
            p_line = space(65)
            p_line = stuff(p_line,col1,len(old_branch),old_branch)
            p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
            p_line = stuff(p_line,col3,len(old_customer),old_customer)
            p_line = stuff(p_line,col5,len(transform(total_1,pic)),transform(total_1,pic))
            p_line = stuff(p_line,col6,1,"*")
            DO Prt_Line
      ENDDO
      total_3 = total_3 + total_2
      p_line = space(65)
      p_line = stuff(p_line,col1,len(old_branch),old_branch)
      p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
      p_line = stuff(p_line,col5,len(transform(total_2,pic)),transform(total_2,pic))
      p_line = stuff(p_line,col6,2,"**")
      DO Prt_Line
   ENDDO
   grand_total = grand_total + total_3
   p_line = space(65)
   p_line = stuff(p_line,col1,len(old_branch),old_branch)
   p_line = stuff(p_line,col5,len(transform(total_3,pic)),transform(total_3,pic))
   p_line = stuff(p_line,col6,3,"***")
   DO Prt_Line
ENDDO
p_line = space(65)
p_line = stuff(p_line,20,11,"GRAND TOTAL")
p_line = stuff(p_line,col5,len(transform(grand_total,pic)),transform(grand_total,pic))
DO Prt_Line
close database
set procedure to
set talk on
return
```

Figure 3: 1st solution for the 3-level control break problem using 4 nested DO loops

The purpose of all solutions is to take the sorted (or indexed) invoice table and produce the output as illustrated in Figure 1. As specified, a detail line is required for each invoice. Summary lines are required for each branch, salesman, and customer. Another summary line is required for the overall grand total. Consequently, the logic is more involved than simply obtaining an invoice record, making a few calculations, and printing the results.

A common subroutine, which is stored in '*cbreak*' is called for '*initialization*'. Four DO loops are controlling execution of the program. Note that the outmost DO loop covers all records in the invoice table. The innermost DO loop summarizes and prints all sales amounts for a given customer number.

Studying the structure of the source code quickly reveals how it could by generalized to handle more than 3 levels of totals. In general (n+1) DO loops are required for solving a n-level control break problem.

```
set talk off
clear
set procedure to cbreak
do initialization
store 0 to total_1, total_2, total_3
use invoice.dbf
index on branch+ salesman + customer + invoice tag combi_key
go top
old_branch   = branch
old_salesman = salesman
old_customer = customer
DO while .not. eof()
   IF branch > old_branch
      DO customer_change
      DO salesman_change
      DO branch_change
   ELSE
      IF salesman > old_salesman
         DO customer_change
         DO salesman_change
      ELSE
         IF customer > old_customer
            DO customer_change
         ENDIF
      ENDIF
   ENDIF
   DO process_detail
ENDDO
DO customer_change
DO salesman_change
DO branch_change
p_line = space(65)
p_line = stuff(p_line,20,11,"GRAND TOTAL")
p_line = stuff(p_line,col5,len(transform(grand_total,pic)),transform(grand_total,pic))
DO Prt_Line
close database
set procedure to
set talk on
return
```

Figure 4: 2nd Solution for the 3-level control break problem using a modular program design

A significant new element in the 2nd solution compared to the previous one is, of course, the frequent use of subroutines called by corresponding DO statements. As programs become larger in size and complexity, it becomes increasingly necessary to standardize and organize the programming approach. It becomes more essential to segment the program for purposes of clarity.

Although the '*segments*' will be written as separate routines, they do not precisely fall within the general definition of a subroutine, that is, a routine that is used by different parts of the program. This is usually not the case in modular programming. The program is subdivided into these segments mainly for the purpose of clarity. It has no particular advantage in the use of memory. It does, however, have the following advantages:

- Thinking is organized around the main structure of the program. The various more complicated but less relevant routines are kept separate.
- subroutines may be developed and debugged separately.
- It permits easier legibility for others who subsequently will have to read and understand the program

```
procedure branch_change
grand_total = grand_total + total_3
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col5,len(transform(total_3,pic)),transform(total_3,pic))
p_line = stuff(p_line,col6,3,"***")
DO Prt_Line
store 0 to total_3
old_branch = branch
return

procedure salesman_change
total_3 = total_3 + total_2
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col5,len(transform(total_2,pic)),transform(total_2,pic))
p_line = stuff(p_line,col6,2,"**")
DO Prt_Line
store 0 to total_2
old_salesman = salesman
return

procedure customer_change
total_2 = total_2 + total_1
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col3,len(old_customer),old_customer)
p_line = stuff(p_line,col5,len(transform(total_1,pic)),transform(total_1,pic))
p_line = stuff(p_line,col6,1,"*")
DO Prt_Line
store 0 to total_1
old_customer = customer
return
```

Figure 4: Continued

```
set talk off
clear
set procedure to cbreak
DO initialization
use invoice.dbf
index on branch+ salesman + customer + invoice tag combi_key
go top
store 0 to total_1, total_2, total_3
old_branch   = branch
old_salesman = salesman
old_customer = customer
DO while .not. eof()
   DO CASE
      CASE branch > old_branch
         DO branch_change
      CASE salesman > old_salesman
         DO salesman_change
      CASE customer > old_customer
         DO customer_change
   ENDCASE
   DO process_detail
ENDDO
DO branch_change
p_line = space(65)
p_line = stuff(p_line,20,11,"GRAND TOTAL")
p_line = stuff(p_line,col5,len(transform(grand_total,pic)),transform(grand_total,pic))
DO Prt_Line
close database
set procedure to
set talk on
return
```

Figure 5: 3rd solution for the 3-level control break problem using cascading procedures

Also the 3rd solution handles control breaks as subroutines. As opposed to the previous solution control break routines now call lower-level routines. For example, the BRANCH_CHANGE routine calls the SALES_CHANGE routine which, in turn, invokes the CUSTOMER_CHANGE routine.

A second difference arises from substitution of 3 nested IF statements by a single CASE statement.

```
procedure branch_change
DO salesman_change
grand_total = grand_total + total_3
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col5,len(transform(total_3,pic)),transform(total_3,pic))
p_line = stuff(p_line,col6,3,"***")
DO Prt_Line
store 0 to total_3
old_branch = branch
return

procedure salesman_change
DO customer_change
total_3 = total_3 + total_2
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col5,len(transform(total_2,pic)),transform(total_2,pic))
p_line = stuff(p_line,col6,2,"**")
DO Prt_Line
store 0 to total_2
old_salesman = salesman
return

procedure customer_change
total_2 = total_2 + total_1
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col3,len(old_customer),old_customer)
p_line = stuff(p_line,col5,len(transform(total_1,pic)),transform(total_1,pic))
p_line = stuff(p_line,col6,1,"*")
DO Prt_Line
store 0 to total_1
old_customer = customer
return
```

Figure 5: Continued

```
set talk off
clear
set procedure to cbreak
DO initialization
store .f. to branch_flag, salesman_flag
store  0  to total_1, total_2, total_3
use invoice.dbf
index on branch+ salesman + customer + invoice tag combi_key
go top
old_branch   = branch
old_salesman = salesman
old_customer = customer
DO while .not. eof()
   IF branch > old_branch
      store .t. to branch_flag, salesman_flag
      DO customer_change
   ELSE
      IF salesman > old_salesman
         store .t. to salesman_flag
         DO customer_change
      ELSE
         IF customer > old_customer
            DO customer_change
         ENDIF
      ENDIF
   ENDIF
   DO process_detail
ENDDO
store .t. to branch_flag, salesman_flag
DO customer_change
p_line = space(65)
p_line = stuff(p_line,20,11,"GRAND TOTAL")
p_line = stuff(p_line,col5,len(transform(grand_total,pic)),transform(grand_total,pic))
DO Prt_Line
close database
set procedure to
set talk on
return
```

Figure 6: 4th solution for the 3-level control break problem using 2 switches

This time, a switch is turned on to record the fact, that a particular level of control break has occurred. Printing of totals can proceed starting at the lowest level. The state of the switches is used to decide at which level to stop printing of totals.

Switches can be very useful when used in moderation. However, the use of numerous switches in a program can make it very confusing and difficult to follow and should be avoided if possible.

```
procedure branch_change
grand_total = grand_total + total_3
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col5,len(transform(total_3,pic)),transform(total_3,pic))
p_line = stuff(p_line,col6,3,"***")
DO Prt_Line
store 0 to total_3
old_branch = branch
store .f. to branch_flag
return

procedure salesman_change
total_3 = total_3 + total_2
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col5,len(transform(total_2,pic)),transform(total_2,pic))
p_line = stuff(p_line,col6,2,"**")
DO Prt_Line
store 0 to total_2
IF branch_flag
   DO branch_change
ENDIF
old_salesman = salesman
store .f. to salesman_flag
return

procedure customer_change
total_2 = total_2 + total_1
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col3,len(old_customer),old_customer)
p_line = stuff(p_line,col5,len(transform(total_1,pic)),transform(total_1,pic))
p_line = stuff(p_line,col6,1,"*")
DO Prt_Line
store 0 to total_1
IF salesman_flag
   DO salesman_change
ENDIF
old_customer = customer
return
```

Figure 6: Continued

```
set talk off
clear
set procedure to cbreak
DO initialization
use invoice.dbf
index on branch+ salesman + customer + invoice tag combi_key
go top
DO while .not. eof()
   DO branch_change
ENDDO
p_line = space(65)
p_line = stuff(p_line,20,11,"GRAND TOTAL")
p_line = stuff(p_line,col5,len(transform(grand_total,pic)),transform(grand_total,pic))
DO Prt_Line
close database
set procedure to
set talk on
return
```

Figure 7: 5th solution for the 3-level control break problem using a strict top-down approach

Four DO loops are used to control the flow of the 5th solution. In this respect it is similar to the 1st solution. The distinction comes not only from handling the control breaks as subroutines. In addition each of them contains a DO loop that drives the next lower-level control break routine. For example, routine BRANCH_CHANGE drives the subordinate routine SALESMAN_CHANGE. This routine calls its lower-level routine CUSTOMER_CHANGE. This subroutine, in turn, calls its subordinate routine PROCESS_DETAIL. Thus, all control break subroutines are linked together in a strict top-down manner.

```
procedure branch_change
old_branch = branch
total_3 = 0
DO while old_branch = branch
   DO salesman_change
ENDDO
grand_total = grand_total + total_3
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col5,len(transform(total_3,pic)),transform(total_3,pic))
p_line = stuff(p_line,col6,3,"***")
DO Prt_Line
return

procedure salesman_change
old_salesman = salesman
total_2 = 0
DO while old_salesman = salesman
   DO customer_change
ENDDO
total_3 = total_3 + total_2
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col5,len(transform(total_2,pic)),transform(total_2,pic))
p_line = stuff(p_line,col6,2,"**")
DO Prt_Line
return

procedure customer_change
old_customer = customer
total_1 = 0
DO while old_customer = customer .and. .not. eof()
   DO process_detail
ENDDO
total_2 = total_2 + total_1
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col3,len(old_customer),old_customer)
p_line = stuff(p_line,col5,len(transform(total_1,pic)),transform(total_1,pic))
p_line = stuff(p_line,col6,1,"*")
DO Prt_Line
return
```

Figure 7: Continued

```
procedure initialization
public pic, col1, col2, col3, col4, col5, col6
public line_limit, line_cnt, page_cnt, grand_total
pic = "##,###,###,###.##" && picture definition for column 5
col1        =  0 && start of branch column
col2        = 10 && start of salesman column
col3        = 20 && start of customer column
col4        = 30 && start of invoice column
col5        = 40 && start of (total) sales amount column
col6        = 58 && start of control break indicator column
line_limit  = 50 && number of pint lines per page
line_cnt    = 99 && print line counter
page_cnt    =  0 && page counter
grand_total =  0 && grand total
return

procedure process_detail
total_1 = total_1 + sales
p_line = space(65)
p_line = stuff(p_line,col1,len(old_branch),old_branch)
p_line = stuff(p_line,col2,len(old_salesman),old_salesman)
p_line = stuff(p_line,col3,len(old_customer),old_customer)
p_line = stuff(p_line,col4,len(invoice),invoice)
p_line = stuff(p_line,col5,len(transform(sales,pic)),transform(sales,pic))
DO Prt_Line
skip
return

procedure Prt_Line
IF line_cnt >= line_limit
   DO New_Page
ENDIF
? p_line at 0 font 'courier'
line_cnt = line_cnt + 1
return

procedure New_Page
page_cnt = page_cnt + 1
?  "SALES REPORT"  at   25 font 'courier'
?? "PAGE"          at   56 font 'courier'
?? str(page_cnt,3) at col6 font 'courier'
?
?  "BRANCH"        at col1 font 'courier'
?? "SALESMAN"      at col2 font 'courier'
?? "CUSTOMER"      at col3 font 'courier'
?? "INVOICE"       at col4 font 'courier'
?? "SALES AMOUNT"  at 46   font 'courier'
line_cnt = 3
return
```

Figure 8: Common procedures

There are 4 common procedures. They are used by all 5 solutions for:

- initialization,
- detail processing,
- printing of detail and summary lines,
- testing of page overflow.

Due to the regularity of the n-level control break problem it is rather simple to write a program that generates corresponding source code.