How To Sort a Data Structure on One or more Subfields (in memory)

Tech tip courtesy of Barsa Consulting, LLC and Dave Schnee

One of the things we used to wish for was a way to sort several arrays according to the data in one or more of them. When there were only 2 arrays and one of them was to be the sort "key", then it was relatively easy to use the SORTA verb in RPG; any other combination required some creativity. More generally, we now have multi-occurrence data structures and data structures based on a pointer either of which may have a variable number of populated entries at any moment in time. Well, using the 'qsort' library function from the C library, an RPGLE program can sort a memory structure according to any desired arbitrary combination of "key" fields. An example is shown below of a complete program that uses qsort and provides the required compare routine itself. Of course, the compare routine could be provided as a separate module and qsort can be used by yet another module of an ILE program.

```
h
 *  Compile with DFTACTGRP(*no) BNDDIR(qc2le)

 *   Prototype of the C Library 'qsort' function
d qsort           pr                  extproc('qsort')
d  ArrayToSort                  *   value
d  #Elements                   10u 0 value
d  ElementSize                 10u 0 value
d  CompareRoutine...
d                                *   procptr value

d MyArray         ds
d   a                            5a   inz('ABCDE')
d   b                            5a   inz('XYZEF')
d   c                            5a   inz('ABAAA')
d   d                            5a   inz('GBCDE')

 *  Note:  "MyArray" could be a DS based on a pointer
 *         and, at run time, ALLOC space to that pointer
 *         equal to the number of occurrences desired times
 *         the size of one element.  If more entries are needed
 *         later, REALLOC the pointer to the new size.

d  #Elements      s             10u 0 inz(4)
d  ElementSize    s             10u 0 inz(5)

d  CompareRoutine...
d                 s               *   procptr
d                                     inz(%paddr('COMPRTN'))

 *   Prototype of the "compare" routine (below)
d CompRtn         pr             10i 0
d  Element1                      *   value
d  Element2                      *   value

 *   Sort the multi-occurrence data structure.
c                   callp     qsort( %addr(MyArray) :
c                                    #Elements      :
c                                    ElementSize    :
c                                    CompareRoutine )

 *   Display the resulting structure.
c     a             dsply
c     b             dsply
c     c             dsply
c     d             dsply
```

```
c                   eval      *inlr = *on
c                   return

 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *
 *  Compare routine.  Returns a -1, 0 or +1 based upon
 *      the relative values of bytes 2-4 of the two
 *      5-byte elements passed in.
 *
 *   Result       Meaning ................
 *     0          Values are to be considered EQUAL
 *    -1          First values is *LT the second value
 *    +1          First values is *GT the second value
 *
 *   (Reverse the +1/-1 to sort DESCENDING on this subfield)
 *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

p CompRtn         b                   export

d CompRtn         pr            10i 0
d  Element1                       *   value
d  Element2                       *   value

d CompRtn         pi            10i 0
d  Element1                       *   value
d  Element2                       *   value

d Value1          ds                  based(Element1)
d  Filler1a                        1a
d  SortOn1                         3a
d  Filler1b                        1a

d Value2          ds                  based(Element2)
d  Filler2a                        1a
d  SortOn2                         3a
d  Filler2b                        1a

d Result          s             10i 0

 *   Assume they are equal (result = *zero)
c                   eval      Result = *zero

c                   if        SortOn1 > SortOn2
c                   eval      Result = 1
c                   endif

c                   if        SortOn1 < SortOn2
c                   eval      Result = -1
c                   endif

c                   return    Result

p CompRtn         e
```