How to Sort "MacMumps" and Other Strange Guys

by Winfried Gerum

Winfried Gerum

Sorting and searching is a big topic for programmers using languages other than MUMPS. For us M folks it is not an issue. M does many things for us. In particular, it automatically sorts data entries according to a very powerful scheme: numeric values sort according to their numeric value, and all other keys sort according to the values of their (ASCII) codes. In many cases that is a perfect solution.

The Good Old Days

In the early days of working with computers, one could forget about accents and umlauts—not even lowercase characters were available. In those days people were happy to see a few words between long rows of numbers. You would get the following set of names sorted as MACARIOS, MACMUMPS, OESTERREICHER, VONNEUMANN, ZAPATA and nobody would complain: M performed exactly as one would expect.

More Functionality—More Problems

Computers became more affordable and people became more demanding: lowercase characters were introduced to the computing world. Using MUMPS and ASCII collating, the same guys sort as:

MacMumps, Macarios, Oesterreicher, Zapata, vonNeumann

The problem is that the ASCII codes of all lowercase characters are higher than `$ASCII("Z")`. MacMumps collates before Macarios because *a* sorts after *M*. Likewise, vonNeumann comes after Zapata because a *v* sorts after *Z*.

It Works as Designed (It Is Not Designed to Work)

When an end user reported this as an "error," someone told him, "That cannot be changed, that is the way your computer works." Besides, it was not a serious issue, as MacMumps and vonNeumann are rare exceptions. Most names follow the pattern 1U1.L, resulting in perfectly reasonable sorting. The occasional exceptions could be explained to the few inquiring customers.

Standardization—The Neverending Story

But progress has a dynamic of its own. Reality soon exceeded the dream of standardization: to accommodate various needs the basic ASCII character set has been extended by different people in many different ways. There now are many different and incompatible national, international, and proprietary character sets. While it now may be possible to use umlaut characters, you may get, depending on the character set and the M implementations, the following sequences of our guys in the sample:

MacMumps, Macarios, Zapata, Österreicher, vonNeumann
MacMumps, Macarios, Zapata, vonNeumann, Österreicher
Österreicher, MacMumps, Macarios, Zapata, vonNeumann

The first line uses a German national character set. Some of the (7-bit) ASCII characters are replaced by umlauted characters. *Ä, Ö, Ü* come between Z and *a*. The second and third lines may actually result from use of the (same) 8-bit character set, but from two different M implementations: some sort 8-bit characters after `$C(127)`, some before `$C(0)`.

Accept the Challenge!

It looks confusing, and it is. We can no longer force people to accept that confusion. They won't accept it and they don't have to. Computers do (most of the time!) exactly as we tell them. We just have to tell computers a little bit more. With some additional care one can get any desired collation sequence very easily. Obtaining the desired sorting requires a transform. Unless that transform is reversible, the full original reference also must be stored somewhere. This is not as easy as `$TRANSLATE`-ing one character into another character, which might be sufficient to accommodate accents. In the Spanish language, the combination *ll* does not sort as two characters but as one. The umlauts and the sharp *s* of the German language sort as two characters (in phone-book collating, not in Duden collating).

The Solution

I have devised a generic algorithm that helps to solve many kinds of sorting problems. That algorithm is table-driven. You supply a table that simply specifies "Character X sorts like U". That table and my algorithm will transform input strings to a unique string with the desired collation property. There is also a backward transform to regain the original string from the transformed one. The basic idea is to replace all characters with their correctly sorting cousins, yielding a string that looks suspiciously like those of the old halcyon days. To distinguish between different strings that yield the same primary sequence, a second sequence of differentiating characters is calculated and appended to the primary sequence. These differentiating characters are needed to make the transform reversible and they distinguish between different keys yielding the same primary sequence. Thus the length of a key is about doubled. This extension is the price that must be paid for the convenience of having a reasonable sorting. (Did you expect a free lunch?). Clever application of the algorithm, however, makes it possible to reduce the average additional space requirements dramatically, at least for some cases.

A character not in the table represents itself. As a differentiating number it is assigned a zero. If a character is in the table, the longest matching entry is searched. The differentiating number is determined by the position of the replaced string (i.e., its first character) within the table. The ordering of entries within the table is significant, allowing fine tuning of the sorting process. The differentiating numbers are assembled into a string. The starting position (or offset) again is user-defined. It should have a very small code value, in order to minimize its effect on collating. If control characters are permitted, the offset may be zero. Otherwise an offset of 32 is recommended. If you want to "read" the transformed keys, an offset of 48 is convenient, as it maps the first ten numbers into their ASCII representation. To reduce storage requirements, it is possible to separate the primary transform and the differentiating string by a delimiter character. The ASCII value of this character also should be very small and it is essential for the table to specify an entry that maps this character onto another one. The secondary string is then trimmed of trailing zeroes (or the equivalent). If all characters of a string are mapped into themselves, the transform changes nothing and no additional space is required.

Our sample names are converted by `$$UIN^%GKT()` to

```
MACARIOS !!!!!!!
MACMUMPS !! !!!!
OESTERREICHER ""!!!!!!!!!!!
VONNEUMANN !!! !!!!!!
ZAPATA !!!!!
```

These keys sort as we would expect them to. With $$UOUT^%GKT() we get the original strings back:

Macarios, MacMUMPS, Österreicher, vonNeumann, Zapata

Armed with this function, sorting and searching, once again, is no longer a *big* issue for M programmers. As we have seen, user-defined collation is available with existing M implementaitons. Under the heading of Internatinalization, the MUMPS Development Committee has done considerable work. One of the new miracles is the feature of a structured system variable (<u>ssvn</u>) describing collation properties:

`^$C(`<u>`charsetexpr`</u>`,"COLLATE")=`<u>`expr`</u> <u>`V`</u> <u>`algoref`</u>

The idea is, that before M collation is done, strings are internally transformed by a function described in this ssvn. The generic key tranform described in this article produces exactly the collation value that the `charset algoref` function should produce. Once the new standard is generally available, it is possible to specify this transform in one place with the benefit of getting more reasonable collation without the burden of changing existing applications.

The routine is shown in the following figure.

Winfried Gerum is president of Winner Software in Erlangen, Germany. His column appears regularly in *M Computing*.

```
%GKT   ;generic key transform;
   +1  ;Gerum,20-March-1992
   +2  Q
   +3  ;function $$IN^%GKT(X,T[,O[,D]])
   +4  ;in:X string to be transformed
   +5  ; T transformation table (string)
   +6  ; 1st char usually ":" delimiter within replacement pair
   +7  ; 2nd char usually "," delimiter between replacement pairs
   +8  ; following chars are pairs like
   +9  ; " :_,a:A,A:A,b:B,Ä:AE,sch:S"
   +10 ; meaning "a" sorts as "A","b" as "B", "Ä" as "AE" etc.
   +11 ; both parts of a replacement pair should be at least one
   +12 ; character long. If you specify "-" sorts like no character,
   +13 ; the reverse transform cannot recover the "-" character.
   +14 ; If you specify "" sorts like some character, that is simply
   +15 ; ignored. So the delimiting chars themselves cannot be
   +16 ; transformed. The order within the table is significant: if
   +17 ; several combinations of chars map into the same char the
   +18 ; order within the table decides which one sorts first. Each
   +19 ; character not in the table sorts before occurrences of the
   +20 ; same character resulting from a transform.
   +21 ; O offset for counts string, recommended value O if CTRLs are
   +22 ; permitted as subscripts, otherwise 32 (=$ASCII(" "))
   +23 ; default value is 48 (=$ASCII("0"))
   +24 ; if called by name, value actually used is returned
   +25 ; D delimiter between raw transform and counts string, should be
   +26 ; zero or one character. Surplus chars are being discarded.
   +27 ; If non-empty that character either should never be in an input
   +28 ; string or it should appear in the translation table. Use of the
   +29 ; delimiter saves space in case no characters of the input string
   +30 ; are translated.
   +31 ; if it is not empty, its $ASCII value should be small.
   +32 ; default value is ""
   +33 ; if called by name, value actually used is returned
   +34 ;out: transformed key, with desired collation properties
IN  (X,T,O,D)N B,C,F,I,P,Q,Y,Z
   +1  Q:$G(X)="" "" ;missing or trivial string
   +2  Q:$L($G(T))<5 X ;missing or trivial translation table
   +3  S:$G(O)="" O=48 ;offset defaults to $ASCII("0")
   +4  S D=$E($G(D)) ;delimiter defaults to empty
   +5  S P=$E(T),Q=$E(T,2)
   +6  S Y="",Z=""
   +7  F I=1:1:$L(X) D S Y=Y_C,Z=Z_B
   +8  .S C=$E(X,I) I T'[(Q_C) S B=$C(0) Q ;no transform
   +9  .F I=I:1:$L(X) Q:T'[(Q_C_ $E(X,I+1)) Q:$E(X,I+1)=P S C=C_ $E(X,I+1) ;get longest matching entry
   +10 .I T'[(Q_C_P) S B=$C(0) Q
   +11 .S F=$P($P(T,Q_C_P,2),Q) ;get replacement
   +12 .S B=$L($P(T_Q,Q_C_P),P_ $E(F)) ;get of entry
   +13 .S C=F,B=$TR($J("",$L(C))," ",$C(O+B))
   +14 I D]"" F I=$L(Z):-1:0 I $A(Z,I)'=0 S Z=$E(Z,1,I) Q
   +15 I D]"",Z="" Q Y ;transform yields trivial result
   +16 Q Y D Z
   +17 ;function $$OUT^%GKT(X,T[,O[,D]])
   +18 ;reverse function to IN, in and out same as above
OUT (X,T,O,D)N B,C,c,F,I,P,Q,Y,Z
   +1  Q:$G(X)="" "" ;missing or trivial string
   +2  Q:$L($G(T))<5 X ;missing or trivial translation table
   +3  S:$G(O)="" O=48 ;offset defaults to $ASCII("0")
   +4  S D=$E($G(D)) ;delimiter defaults to empty
   +5  S P=$E(T),Q=$E(T,2)
   +6  I D]"" S Y=$P(X,D),Z=$P(X,D,2,$L(X,D)) Q:Z="" Y
   +7  E S I=$L(X),Y=$E(X,1,I/2),Z=$E(X,I/2+1,I)
   +8  S F=""
   +9  F I=1:1:$L(Y) S C=$E(Y,I) D S F=F_C
   +10 .S B=$E(Z,I) Q:B="" S B=$A(B)-O Q:'B ;no replacement
   +11 .S C=$L($P(T,P_C,1,B),Q) ;Piece # of replacement pair
   +12 .S C=$P(T,Q,C) ; replacement pair
   +13 .S c=$P(C,P,2) ;replacing string
   +14 .S C=$P(C,P) ;original string
   +15 .I C=""!(c="") S F="<inconsistent input>",I=$L(Y) Q
   +16 .S I=I+$L(c)-1 ;skip (length of replacement)
   +17 Q F
   +18 ;function $$OUT2^%GKT(X,TI,TO[,O[,D]])
   +19 ;reverse function to $$IN, similar to $$OUT. The difference is that
   +20 ;in this generalized form there are two occurrences of the translation
   +21 ;table. One (TI) as it has been used to transform the original string,
   +22 ;the other (TO) is an alternate translation table that would be used
   +23 ;by $$IN to generate the same output as TI for a different character set.
   +24 ;thus it is possible to have a common exchange format between different
   +25 ;character set. The mechanism may be used to produce sequences for a
   +26 ;printer to emulate characters by things like
   +27 ;<CHAR1><BACKSPACE><CHAR2>.
```

```
OUT2(X,TI,TO,O,D)N B,C,c,F,I,P,Q,Y,Z
   +1  Q:$G(X)="" "" ;missing or trivial string
   +2  Q:$L($G(TI))<5 X Q:$L($G(TO))<5 X ;missing or trivial translation table
   +3  S:$G(O)="" O=48 ;offset defaults to $ASCII("0")
   +4  S D=$E($G(D)) ;delimiter defaults to empty
   +5  ;DEBUG; Q:$$TITO(TI,TO) "<incompatible rules>"
   +6  S P=$E(TI),Q=$E(TI,2)
   +7  I D]"" S Y=$P(X,D),Z=$P(X,D,2,$L(X,D)) Q:Z="" Y
   +8  E S I=$L(X),Y=$E(X,1,I/2),Z=$E(X,I/2+1,I)
   +9  S F=""
   +10 F I=1:1:$L(Y) S C=$E(Y,I) D S F=F_C
   +11 .S B=$A(Z,I)-O Q:'B ;no replacement
   +12 .S C=$L($P(TI,P_C,1,B),Q) ;Piece # of replacement pair
   +13 .S c=$P($P(TI,Q,C),P,2) ;replacing string
   +14 .S C=$P($P(TO,Q,C),P) ;original string
   +15 .I C=""!(c="") S F="<inconsistent input>",I=$L(Y) Q
   +16 .S I=I+$L(c)-1 ;skip (length of replacement)
   +17 Q F
   +18 ;function to test whether parameters TI,TO for $$OUT2 are compatible
   +19 ;returns TRUE, if not compatible
TITO(TI,TO)     N I,P,Q S P=$E(TI),Q=$E(TI,2)
   +1  I $E(TI,1,2)'=$E(TO,1,2)
   +2  E I $L(TI,P)'=$L(TO,P)
   +3  E I $L(TI,Q)'=$L(TO,Q)
   +4  E F I=2:1:$L(TI,Q) I $P($P(TI,Q,I),P,2)'=$P($P(TO,Q,I),P,2) Q
   +5  Q $T
   +6  ;sort all lowercase as uppercase
UIN(X) Q $$IN(.X,":,:!,!:!,a:A,b:B,c:C,d:D,e:E,f:F,g:G,h:H,i:I,j:J,k:K,l:L,m:M,n:N,o:O,p:P,q:Q,r:R,
     s:S,t:T,u:U,v:V,w:W,x:X,y:Y,z:Z,Ä:AE,ä:AE,Ö:OE,ö:OE,Ü:UE,ß:SS",32," ")
   +1 ;inverse of $$UIN
UOUT(X) Q $$OUT(.X,":, :!,!:!,a:A,b:B,c:C,d:D,e:E,f:F,g:G,h:H,i:I,j:J,k:K,l:L,m:M,n:N,o:O,p:P,q:Q,
      r:R,s:S,t:T,u:U,v:V,:W,x:X,y:Y,z:Z,Ä:AE,ä:AE,Ö:OE,ö:AE,Ü:UE,ß:SS",32," ")
```

Figure 1. The routine for sorting.

MTA MEMBERSHIP APPLICATION

☐ **YES, I want to enhance my professional career. Sign me up as a member today!**

PLEASE CHECK APPROPRIATE BOX

☐ Individual — **$75** ☐ Individual, Non-North American — **$85**

☐ Organizational—(choose one) ☐ Corporate—**$550** ☐ Institutional [501(C)3]—**$500**

☐ Employee of Organizational Member — **$60** Organizational ID #________________

☐ Student — **$30** **Full-time students only. Attach a copy of current student ID.**

PAYMENT OPTIONS

☐ Check enclosed ☐ Bill Me — Purchase Order # ________________________

☐ Visa ☐ MasterCard — Account # ________________________ Exp. date_____/_____/_____

Name on card ________________________________ Signed ________________________________

For your convenience, you may fax or send this form to:

M Technology Association
1738 Elton Road, Suite 205
Silver Spring, MD 20903-1725
Fax: 301-431-0017

MCI3