This bulletin describes the changes made to the user interfaces of `sort_items_` and `sort_items_indirect_`. These procedures provide a highly efficient, yet generalized, sorting facility.

The procedures implement adaptations of the QUICKERSORT algorithm of M. H. van Emde; algorithm A402; Comm. ACM; Vol 13; No 11; Nov, 1970; pp 693-4. A description of the algorithm may be found in Comm. ACM; Vol 13; No 9; Sep, 1970; pp 563-7. In the case of `sort_items_`, the algorithm was modified to reorder an array of unaligned pointers to the data items, rather than the data items themselves. In the case of `sort_items_indirect_`, an array of indices into the pointer array is reordered. In both cases, the algorithm has been made non-recursive.

The procedures incorporate the modification to detect ordered sequences, as suggested by Robert E. Wheeler in his Remark on Algorithm 402 [M1].

Please address all comments or suggestions to:

Ross E. Klinger  
MIT Bldg. 39, Rm. 584  x3224

Or, send mail to: Klinger.PDO

Multics Project Internal working documentation. Not to be reproduced or distributed outside the Multics Project.
The calling sequences have been changed to minimize unnecessary argument manipulation caused by alignment considerations. The arguments used are as follows:

- **vp**
  - (Input). A pointer to a structure containing an array of pointers to the data items to be sorted.

- **ip**
  - (Input). A pointer to the structure into which the ordered array of indices to the pointer array will be placed.

- **ip**
  - (Input). A pointer to a structure containing the lengths of adjustable string data items to be sorted.

- **length**
  - (Input). The length of fixed-length string data items to be sorted; a fixed bin (24) number.

- **function**
  - (Input). An entry variable used to call a user-supplied function which can determine the comparative relation between two data items of arbitrary format.

The structure pointed to by vp should be declared as follows, where n is the value of v.n:

```c
declare 1 v aligned,
   2 n fixed bin (24),
   2 vector (n) ptr unaligned;
```

The structure pointed to by ip or ip should be declared as follows, where n is the value of a.n:

```c
declare 1 a aligned,
   2 n fixed bin (24),
   2 array (n) fixed bin (24);
```

The function should be defined as follows:

```c
dcl function entry (ptr unal, ptr unal) returns (fixed bin (1));
```

The pointers refer to data items 1 and 2, respectively.

- If `data_item_1 < data_item_2`, the function should return -1.
- If `data_item_1 = data_item_2`, the function should return 0.
- If `data_item_1 > data_item_2`, the function should return +1.
The following entry points are defined:

For fixed-length unaligned bit strings:

\[
\text{sort_items}_{\text{Sbit}} (vP, \text{length}); \quad \text{sort_items}_{\text{Indirect_Sbit}} (vP, IP, \text{length});
\]

For fixed-length unaligned character strings:

\[
\text{sort_items}_{\text{Schar}} (vP, \text{length}); \quad \text{sort_items}_{\text{Indirect_Schar}} (vP, IP, \text{length});
\]

For fixed bin (35, 0) numbers:

\[
\text{sort_items}_{\text{Sfixed_bin}} (vP); \quad \text{sort_items}_{\text{Indirect_Sfixed_bin}} (vP, IP);
\]

For float bin (63) numbers:

\[
\text{sort_items}_{\text{Sfloat_bin}} (vP); \quad \text{sort_items}_{\text{Indirect_Sfloat_bin}} (vP, IP);
\]

For data items of arbitrary format:

\[
\text{sort_items}_{\text{Sgeneral}} (vP, \text{function}); \quad \text{sort_items}_{\text{Indirect_Sgeneral}} (vP, IP, \text{function});
\]

For adjustable length character strings:

\[
\text{sort_items}_{\text{Indirect_Sadj_char}} (vP, IP, IP); \quad \text{(not implemented with sort_items.)}
\]