Computers and the Humanities 18 (1984)

bibliofile: Humanizing the UNIX System*

Earl H. Kinmonth

UNIX, developed by Bell Laboratories and widely available at American colleges and universities,[1] is generally low cost, provides extensive text processing software and overall user-friendliness, and is therefore a particularly attractive system for humanists. This attraction is sure to increase with the recent announcement that IBM will be offering **UNIX** on its small machines. Nevertheless, there are many rough spots in this operating system. Its data base (**dbminit**) is expensive because of its disk demands and it provides only the most rudimentary functions.[2] Programs developed by the author to give **UNIX** a flexible data base follow certain principles of program design and implementation that imply standards by which data base and text processing software in general may be evaluated.

History of bibliofile

The programs described here evolved out of the author's desire for an "electronic shoe box" to store the file cards generated in the course of indexing a book. Finding no such program available at the University of California at Davis, and no administrative interest in acquiring such a program, I wrote several simple programs that have grown into a system which covers all phases of data manipulation. These have attracted heavy usage at UCD, in both individual and institutional bibliographic projects ranging from medieval rhetoric to contemporary Japanese management.[3]

Earl Kinmonth is an associate professor of history at the University of California, Davis (UCD). *This paper and the programs described in it have benefitted substantially from the comments of Dr. Kevin Roddy, Department of Rhetoric, UCD. Funding for program development has been provided by the Teaching Resources Center, the Computer Center, and by the Graduate Research Committee, at UCD.*

The bibliofile Record Structure

That aspect which most distinguishes the **bibliofile** system from commercial software is its open-ended free-format record structure. Each logical record consists of one or more lines, each beginning with an indicator of what is in that line. An empty line separates one logical record from the next. The individual logical records look rather like ordinary file cards (See Fig. 1).

Figure 1.

```
a Kinmonth, Earl H.
v The Self-Made Man in Meiji Japanese Thought:
From Samurai to Salaryman
pub University of California Press
yop 1981
```

The contents of each record is described in a file usually called ".keys" (pronounced "dot keys") although it can have any name the user wishes.[4] Fig. 2 shows a ".keys" designed for the author's **personal** bibliographic work.[5]

Figure 2.

```
a/aut/author
at/art/article title
r/rev/review
s/ser/serial title
vin/vin/volume, issue, or number
d/date/date of publication
v/vt/volume title
vat/vat/volume author ["By ..."]
tr/trans/translator
ved/ved/volume editor
ed/ed/edition number
vnum/vol/volume number
vols/vols/total number of volumes
col/col/collection title
rcol/rcol/romanized collection title
ced/ced/collection or series editor
plp/plp/place of publication
pub/pub/publisher
yop/yop/year of publication
rep/rep/reprint
p/pp/pages
tp/tp/total pages
loc/loc/location
use/use/usage
memo/memo/memo
xref/xref/cross-reference
su/su/subject
```

The first two items on each line are the "short key" and the "long key". The third item is an explanation. For ease in data entry, the "short key" and the "long key" are interchangeable, and may be identical. Keys may be composed of any alphanumeric combination, and the following data may be separated by an item of punctuation or any amount of space (blanks or tabs) that suits the user's fancy or a given terminal. Typing a record into the system is very similar to typing a file card with a typewriter.

No declaration of field length is needed because fields are uniquely defined by the key (tag) and the field separator (a line feed). The only limit on field length is the amount of buffer space available, which is determined by hardware, not the program.[6] There are no type declarations for input; all data are stored as a byte-addressable stream of characters in conformity with the overall **UNIX** file philosophy.[7] Distinctions between different data types (string, numeric, chronological, etc.), are made at the formatting or report-generating stage, not during input.

The advantages to this approach include:

- The resulting record is a good analog to a conventional file card. Even the most naive user can easily understand what is going on.
- The records are largely self-documenting and even a raw dump of the file is intelligible.[8] No special codes will lock up a terminal, cause a line printer to go berserk, or require assembly language translators.[9] At most, the only processing required for printing is to "fold" lines that are too long for the page.[10]
- The record structure is compatible with all **UNIX** utilities which are basically line oriented.[11]
- In the absence of complicated linkages the files are easily transportable, across machines and accounts.[12]
- The records are acceptable even to Jacquard-principle systems.[13] The only procedures necessary are to fold the long lines and to pad all lines to a common length. The **bibliofile** formatters **kform** and **kawk** (described below) can rewrite **bibliofile** records for systems requiring fixed-length fields.

The only disadvantage of this approach is the space taken by the "keys." In the example given in Fig. 1, twelve percent of the entry is accounted for by the "keys" and the following blank. This is, however, a small price to pay for clarity, and is only partially avoided by Jacquard-principle systems. Unless these use some form of marker for unfilled items, they will have more wasted space than the **bibliofile** format. If a marker is used, it will only be marginally more effective than the **bibliofile** format with one or two character keys.[14]

In contrast to such popular systems as dBASE, which uses fixed length fields and pads with blanks, **bibliofile** has no wasted space. More important, especially in humanistic bibliographic work, when the inevitable five- or six-line title occurs, **bibliofile** allocates enough space for that entry without requiring that all records be large enough to handle the worst possible case. Very few data base systems can

efficiently cope with records such as the example in Fig. 3.

Figure 3.

```
a de Guzman, Juan

v Primera parte de la retorica, dividida en catorze combites
de oradores: donde se trata el modo que se deve guardar en
saber seguir un concepto por sus partes, en qualquiera
platica, razonamiento o'sermon, etc.

plp Alcala'de Henares

yop 1589
```

There is no arbitrary truncation with **bibliofile** nor is there any "bleeding".[15]

The **bibliofile** format is also flexible in terms of the number of fields (keys) in any one record. New keys can be added at any time by appending them to the ".keys" or inserting them in it. Space for the keys is allocated dynamically so there is really no upper limit on their number. The order of the keys may be changed at any time and will automatically result in a reordering of the records the next time a program using the keys writes a file.[16]

In practice this flexibility means that the user does not need rigidly defined data structure before beginning input. If an unanticipated category appears, a new field is simply added to the ".keys." If a subset is desired, it is generated by calling the programs with a ".keys" file containing only desired fields.

This flexibility is achieved by an input routine that compares the first "word" in each incoming line to the identifiers in the ".keys." If the "word" is an identifier, a pointer to the line is installed in an array paralleling the keys list. This table of pointers can be read top to bottom, bottom to top, or in any order the user specifies.

Creation of Files

Because the **bibliofile** file structure is so simple, any **UNIX** editor will do for data entry. Nevertheless, the system has its own editor, **ked**, which shares the syntax and commands of the **UNIX** editors **ed** and **ex**.

In the design of **ked**, special attention was given to ease and accuracy of data entry. Outside of the "hard sciences," it is a rare academic computer project in which data entry does not cause more headaches and consume more time and money than the analysis itself. The fewer mistakes made at entry time, the less work later and the lower the overall cost and aggravation.

ked allows fields to be typed in random order. In the example in Fig. 1, "yop" (year of publication) could have been typed before "a" (author). If a field is typed twice, the second occurrence replaces the first; this is the quickest way to make substantial corrections during entry. Since retyping a field may be inadvertent, a warning is generated. Beginning a line with an unknown identifier produces a vocal (bell) protest. Leading and trailing blanks are removed because these are usually the result of sloppy typing or bouncy keyboards.[17] The user can concentrate on the text and not on the screen because the input routine automatically provides carriage returns near the right-hand margin. For the user incapable of remembering keys, **ked** will step through them, asking the user to fill in each item.

All common addressing modes are provided. Records may be called by absolute numerical address or a range of numerical addresses. Patterns composed of literal strings and "wild card" tokens may be used to address records by context. Numerical and pattern-addressing requests can be combined. All records matching or not matching a pattern can be identified.

An extensive set of operators can be applied both at the record (card) and field (key) level. These include substitution, deletion, movement, copying, exchange, numbering, and listing in a variety of formats. Specified groups of records can be selected and written to separate files or to filters (using the **UNIX** pipe mechanism).[18]

Although it is essentially a "line-oriented editor," **ked** emulates a "screen editor" in some respects and it can drive the very powerful "visual" editor **vi** allowing the user to have both **ked's** record addressing capabilities and the convenience of a screen editor.

The syntax for **ked** operations is the same as that used, with some extensions, by the **UNIX** editors **ed** and **ex**. The user can transfer from **ked** to the **UNIX** editors with little effort. Some commercial software does not even maintain a consistent command syntax within the same program let alone over a whole system.[19]

Complex but frequently used commands may also be placed in separate files. Once these commands are working properly, they are used from within **ked** by a statement such as

so abc

where "abc" is the name of the command file. This provision may be coupled with a built-in macro processor to create "natural language" commands for novice users.

ked both tries to spot potential errors and to provide easy recovery from actual errors. For instance, it works on a copy of the file being edited, not the original. Nothing happens to the original unless you request that it be replaced by the edited version. In editing, the last change made can be "undone" whether it affected one record or hundreds. If the system crashes, the original file is untouched, and the copy actually used by **ked**, called a "buffer," remains. Unless there is physical destruction of the disk, input can be resumed from the point of interruption as soon as the system returns to operation.[20] If a user makes changes to a file and tries to quit without saving them, **ked** provides a warning.

All the features described here are technically simple, but are missing from many commercial editors. In fact, **ed**, the standard **UNIX** editor, lacks these elementary capabilities.[21]

In the size of records that can be handled, **ked** goes beyond the **UNIX** editors. **ex**, the more powerful of the **UNIX** line of editors, can handle records of roughly 200,000 total characters, with no one line more than 512 characters. **ked** being open-ended, its addressing tables can be set by the user and the upper limit is set by the hardware, not the program itself.[22] Because the **UNIX** system allows programs to request more core as they execute, there is no good reason for the size of arrays to be compiled-in parameters, as they are in most **UNIX** programs.[23]

Extremely large individual records may also be handled through **bibliofile** by passing the file through **kuso** (use source). This program scans each record for a field (default kuso) that has been designated as containing the names of source files. **kuso** then merges these source files with the **bibliofile** file it is reading to produce a single output file. These source files may be as large as the available hardware permits. **kuso** provides options to control how the merging takes place and whether the original **bibliofile** record appears in the output.

Sorting and Collation

On the level of logical records and individual fields within records, **ked** encourages the random entry of data. In most applications, some type of ordered output is desired. In **bibliofile** ordering and collation is done by **kord**, a program that "drives" the **UNIX** utility **sort**, greatly extending its capabilities and ease of use.[24]

sort is a good example of a program that is well based in computer theory, written with very tight and efficient code, and next to useless because real-life data does not come in the rigidly defined format the program expects. It is machine- but not application-efficient.[25]

sort can deal only with lines. Even if records consisting of more than one line are pasted together to satisfy **sort**, it arbitrarily (and silently!) truncates at 512 characters, a limit that cannot be raised without rewriting the program. The syntax for a sort is messy at best. The program has no provision for missing data and little provision for deviating from a simple-minded a > b or a < b type of collation.

kord extracts a set of lines from **bibliofile** files, pastes these together in the form **sort** expects, writes the script necessary to cause **sort** to work properly, and then rebuilds the input files according to this sorted index. In the process it can alter or filter what is passed to **sort** so that more complex collation can be performed.

kord commands are given in terms of ".keys." For example

s1/aut/d
s2/vt/d

gives a dictionary sort (only letters, digits, blanks are significant) on title within author.

Sorting may follow a hierarchy.

s1/aut|pub/d
s2/at|vt/d

will sort on author or publisher and on article title or volume title, whichever is found first in each specification. Up to eight levels of sorts (limits imposed by **sort**) may be specified.

kord has an "e" option to excise English articles. A "z" option causes empty fields to collate as "zzzzz" rather than as blanks. A "t" option converts dates in standard formats (December 7, 1941; 7 December 1941; or Dec. 7, 1941) into a form (1941:12:07) that will be properly handled by **sort**.

The "t" option is more than a convenience. It encourages the use of natural forms which are less likely to be typed incorrectly in the first place and which are easier to read when the data is being checked for errors. Although the principles presented in standard textbooks on data base management would have one entering dates such as "411207" or something even more cryptic,[26] the principle here is that reducing file size by coding is usually a false economy because it increases the time required to clean and correct the data. What is efficient in machine terms is not necessarily efficient in human or project terms.

The quirks of foreign languages can be handled by providing filters that are called by **kord**. To get rid of French articles so that a title sort will work properly, one creates a script for the **UNIX** editor.[27] This filter might look like Fig. 4.[28]

Figure 4.

```
ex $1 <<x
g/$Le /s//$/
g/$Les /s//$/
g/$La /s//$/
g/$Un /s//$/
g/$Une /s//$/
g/$L'/s//$/
wq
x
```

Assuming this script has been given the name "filter1," from within **kord**, the user merely gives the command

a!filter1 %

After **kord** makes its index, it gives this to "filter1" which strips the French articles.[29] The rewritten index goes to **sort** which does its usual literal ASCII collation, but that works now because the articles have been stripped and these items will no longer collate under "L" or "U." Since only the **index** is changed, nothing is lost from the original data.

kord illustrates another principle of the **bibliofile** system. Commonly used functions (time, article stripping, "z" collation) are built in. Other functions are supported by having an easy mechanism for calling other programs and scripts to supplement the main program. Fortunately, this mechanism is part of the **UNIX** operating system. From within a **C** program, the function call

system ("string")

causes the commands in "string" to be executed as though they were entered from the terminal. At completion, control returns to the calling program.

All **bibliofile** programs designed for interactive use have provision for calling other programs either to perform filter operations or to initiate collateral processes. Most **bibliofile** programs may also be piped: the output from one program is passed to the input of another, or to another **UNIX** facility. Complex sequences of individual programs and their commands may be assembled into scripts. These collections of commands may then be executed by entering the name of the file containing the commands.

Selection

One of the most appropriate uses for a data handling system such as **bibliofile** is to maintain special research bibliographies, slide collections, reprint files, etc. In this use, the major concern is to rapidly extract subsets from the main file or files. In **bibliofile** this is done with **krep**, an analog to the **UNIX grep** series of pattern finders.

krep searches can find patterns as prefixes, suffixes, or included portions of larger patterns or as "words" (patterns bounded by space, punctuation, or beginning and ends of lines). Patterns may have single- or multi-character "wild-cards" and closure (zero or more instances of an element). Alphanumeric range searches are easily specified. All of these operators can be combined with full boolean logic using parentheses to force grouping.

The command syntax is based on that used by **ked** and all **UNIX** pattern matching programs. Two examples, one simple and one complex, indicate the range of possibilities.

/Japan/

will find any records having *Japan, Japanese, Japanology,* etc., anywhere in the record.

/#plp&Tokyo&#y op 193[5-9]/

will find any records indicating place of publication (plp field) as Tokyo and year of publication as any year between 1935 and 1939.

This command syntax may appear cryptic, and to a degree it is. It is, however, shared with a large number of other **UNIX** programs (the # operator to select specific fields is a **bibliofile** extension) and **in practice** proves to be more congenial than something like

find "place of publication" is "Tokyo"
and "year of publication" is "1935-9"

or even

find "plp" is "Tokyo"
and "yop" is "1935-9"

What is easy to understand at first glance is often extremely tedious over the long run and error-prone (because more characters must be typed). Since **bibliofile** need not be demonstrated in a showroom by untrained (to say nothing of incompetent) salesmen, I have been able to program for the serious and repetitive user.

The **krep** logic is optimized so that no unnecessary scanning is required. That is, a failure in a sequence of patterns connected by "and" (&) will stop scanning for that sequence. Similarly, the first success in a sequence connected by "or" (|) also stops the scanning process. By putting the most restrictive (least likely) pattern first in an "and" sequence or the least restrictive (most likely) pattern first in an "or" sequence, the user can substantially reduce the cost of searching long records.

Nevertheless, even with optimization, the linear sweeps of **krep** can be wasteful. Provision is thus made for restricting a search to the first *n*, last *n*, or *m*-to-*n* percent of a file. Given a file sorted by name and a search for works by "Smith," limiting the search to the last half of the file (go50% is the command) will cut costs while retaining the benefits of a linear sweep.

The first version of **kord** provided another alternative to simple linear sweeps: the index used for sorting (typically much smaller than the actual file) could be searched with the program retrieving the original records automatically. The difference in cost and waiting time between linear sweeps and indexed searches was so small, especially in the context of overall project cost, that no users found it worthwhile to sacrifice the ability to examine all fields of a record in order to get somewhat faster response.

In the following examples of actual **krep** (linear) and **kord** (indexed) searches, the test file for these timings was made by repeated copying of a small bibliography used for testing.[30] The resulting file had 157 copies of each record spaced evenly throughout the file for a total of 4082 records. The overall attributes of this file are shown in Fig. 5.

Figure 5.

493451	characters in file
28103	lines in file
98	longest line
20	average line
4082	records in file
227	largest record (chars)
11	largest record (lines)
6	average record (lines)
120	average record (chars)
963	blocks (512 chars)

These parameters are roughly what one can expect from standard English language works in an unannotated bibliography.[31]

Figure 6.

	krep linear		kord indexed	
Pattern	Billed	Real	Billed	Real
Kinmonth	26.8	32	11.6	21
Kinmonth:Tillich	36.3	71	24.4	37

(a) Timings are in seconds and reflect prime-time loading conditions.
(b) The indexing phase in **kord** required 58 seconds billed and 90 seconds real time.
(c) The machine used was a PDP 11/70.
(d) There were 157 records matching "Kinmonth" and 314 matching "Tillich." The time to write these to the output file is included in the chart.

Given the relatively small differentials between indexed and linear searches and the relatively high cost of creating an index, there is very little incentive for giving up the infinite flexibility of the whole-record scan for the highly restrictive indexed method. The linear sweep would be inappropriate for an airline flight reservation system or for putting a whole library system on line. For any file an individual scholar is likely to generate, it is, however, more usable than systems that sacrifice flexibility in the name of machine efficiency. Indeed, after a year of trial, the indexed search function was removed from **kord** because it was not being used.[32] Generally, my experience has been the humanistic users are more interested in flexibility than speed, especially since **bibliofile**, even on a heavily loaded **UNIX** system, can retrieve data faster than can most highly restrictive data bases running on single-user microcomputers.

While ignoring the indexed-search provisions of **kord**, both institutional and humanistic users requested facilities for merging and joining records. **kord** provides three distinct operations for joining two or more separate records that share one or more common data fields into a single unit.

(a) Appending join. If the information in a given field of a second or subsequent card is different from that in the same field on the first card, the resulting card has

key data from 1; data from 2; data from nth

(o) Overlay join. Data from second and subsequent cards replaces that on earlier cards to yield

key data from nth

Figure 7.

XXXVIII *Bibliography*

———. *La Tradition manuscrite des romans de Chrétien de Troyes.* Vol. 2e tirage: Publications romanes et français, 90. Genève: Droz, 1966.

Morawski, Joseph. *Proverbes français antérieurs au XVe siècle.* CFMA, 47. Paris: Champion, 1925.

Neumann, Fritz. *Zur Laut- und Flexionslehre des Altfranzösischen.* Heilbronn: Henninger, 1878.

Nitze, Fritz. "The Character of Gauvain in the Romances of Chrétien de Troyes." *MP*, 50 (1953), 219-25.

Nyrop, K. R. *Grammaire historique de la langue française.* 6 vol. 2nd ed. Paris: Picard, 1925-36.

Omont, H. *Catalogue général des manuscrits français de la Bibliothèque Nationale.* vol. 17. Paris: Leroux, 1900.

P[aris], G[aston]. "Gauvain et Hunbaut," in *Histoire littéraire de la France.* vol 30. Paris: Imprimerie Nationale, 1888.

Perlesvaus. Le Haut Livre du Graal: Perlesvaus. Edited by William A. Nitze and T. Atkinson Jenkins. 2 vol. The Modern Philology Monographs of the University of Chicago. Chicago: University of Chicago Press, 1932-37.

(s) Supplementary join. Blank fields in earlier cards are supplemented by data from later cards which produces

key data from 1

if the first card in a sequence has data for a given key, otherwise

key data from nth

This join capability allows the merging of independently created data sets. It has been used to solve a peculiar problem in medieval bibliography by merging completely separate number-author and number-title lists into a single, far more usable bibliography that can be easily sorted, indexed, and cross-referenced.

Formatting and Printing

For many purposes a simple raw dump of **bibliofile** cards is quite usable. More elaborately formatted output can be prepared with any of the **UNIX** editors. Also suited to this purpose are **sed**, the stream editor, and **awk**, the report generator. **bibliofile** does have its own formatter, something of a cross between **awk** and **sed.**[33] That is, it combines pattern matching and substitution functions with full boolean logic phrased in terms of keys as identifiers and a limited mathematics capability. When used to drive **nroff** or **troff** (the **UNIX** word processing programs), this program (**kform**) can deliver camera ready copy.

kform is programmed in a language similar to that used by **nroff** and **troff.**[34] The language has both primitive elements and complex routines found useful in bibliographic work. Fig. 8 shows a simple program that makes an author-title list. It replaces repeated authors by "————" and inserts appropriate **nroff-troff** instructions for paragraph indentation and boldface type.

Figure 8.

```
.if .no aut || .no vt .th .sk .fi
.if .ep $0,aut .th .pw ".BR$n----------,$n"
.el .pw ".BR$n$s,$n" aut .fi
.sv $0,aut
.pw "%B%s%R.%n" vt
```

Translated this program says:

If there is no author or no volume title, then skip this card.

If there are equal patterns in $0 (previous author) and author, then print and write the **nroff-troff** macro (BR) for a bibliographic entry followed by "————,".

If the patterns are not equal (else case of if-then-else), then print and write the **nroff-troff** macro BR, a linefeed, and the author's name(s) followed by a comma.

Save the current author in register $0 for comparison with the next author.

Print and write the volume title preceded by the **nroff-troff** instructions for **bold** type. Terminate the string with a period and the **nroff-troff** instructions to return to Roman font.

Despite its cryptic commands, **kform** has proved relatively easy to use because, unlike **awk** and **sed,** it is interactive and has full, plain English diagnostics. If the second line in the program had errors, one would get from **kform** the output shown in Fig. 9.

Figure 9.

```
.if .es $0,aut .th .pw ".BR$n----------,$n"
-----!
<unknown function>
.if .ep $0,aut .th .pw ".BR$n----------,$n
------------------------!
<string not terminated>
```

Errors not detected at execution time halt the program with a diagnostic that tells which line in the source program contained the error.

During its compilation phase, **awk** would say "bailing out near line 2," not the most useful debugging aid. Errors during execution simply abort **awk** with a core dump that is, for all practical purposes, useless. For all its error checking **kform** runs faster than **awk.**[35]

While writing scripts for **kform**, one can get a list of commands and their syntax simply by typing "??". Several comprehensive examples are provided in the printed documentation, and

many users have been able to cannibalize or modify these for their own purposes. Because **kform** is interactive, one can test a script one line at a time until the desired results are obtained. Thus, even with the arcane commands, it is possible to develop a working program somewhat more rapidly with **kform** than with batch-style **sed** or **awk**.

Eventually, **kform** will be replaced by a C language interpreter geared to the **bibliofile** system. The initial version of this interpreter provides the following features of the C language: integer and string variables; for, while and if-else control-flow statements; standard arithmetic operators (addition, subtraction, multiplication, division, modulus); and all of the string manipulation functions provided as part of the standard C library.

Using this program, tentatively called **kawk**, the formatting task above is written as in Fig. 10.

Figure 10.

```
char    prevaut[512];
while(rdcard())
{
if($aut && $vt)
    {
    if(strcmp(prevaut,$aut) == 0)
        printf(".BR$n---------,$n");
    else
        printf(".BR$n$s,$n",$aut);
    strcpy(prevaut,$aut);
    printf("%B$n%s.$n",$vt);
    }
}
```

With the exception of the '$' operator to access data fields from **bibliofile** records and extensions to the C format specifications, **kawk** is a well-behaved, interactive subset of the C language with the practical advantage that student assistants drawn from computer science courses of a university using **UNIX** should already be familiar with it. Similarly, the researcher who wants to learn programming will find the highly interactive **kawk** much easier to experiment with than the more powerful (but much slower C compiler) and much more specific in its diagnostics than **awk** (which is also a variation on C).

Learning to Use the System

Programs that emphasize efficient use of computer resources may actually be more expensive in the long run if mechanical efficiency leads to programs that are hard for people to use. For the casual user of a given computer program, the cost of familiarization may well exceed the cost of execution. Thus no special effort has been made to tune **bibliofile** programs for absolute machine efficiency. Instead, once programs are doing what they were intended to do, primary effort has gone into minimizing the learning time and making the inevitable trial-and-error stage reasonably painless and brief.

As far as possible, the command structure of the programs is the same as that for the **UNIX** analog or a logical extension of the analog's commands. In the few instances where a departure was made from **UNIX** practice, either the **bibliofile** program was written to recognize more than one form of the same command or instructions on conversion have been supplied in the documentation. Deviations from **UNIX** patterns have been introduced only when those patterns clearly and consistently invited errors. The basic assumption here is that consistency, even with commands that are more awkward than efficient parsing requires, is preferable to introducing variations that must be memorized or continually checked against documentation.

Although the **UNIX** system is inherently interactive, few of the utilities take advantage of this feature. Most are essentially batch-style programs that might just as well have been written in the 1950s. Effectively used, however, interaction minimizes the learning time even without recourse to menus. If the program catches errors and gives a reasonably clear indication of what is wrong, the user can learn largely by experimenting with output directed to the terminal, adding and changing parameters in response to program output and diagnostic messages. Once the parameters seem to be yielding the proper results, the output can be switched from the terminal to a file or to a

printer. After switching the output, the only instruction that need be repeated is the run command, usually a "go."

Oddly, some otherwise thoughtful programmers, including those associated with the development of **UNIX**, seem to regard diagnostics as something of an electronic VD: a sign that you have strayed from the straight and narrow and are getting your just reward when a program aborts without warning or destroys your data and hours of work. The original **UNIX** editor **ed** allowed one to overwrite files without warning. You could quit without having saved your work. The few errors the program did detect produced only a "?" to indicate a problem.[36] Some **UNIX** utilities give no more than "syntax error" to an immense variety of problems in what are often complex and tedious scripts.[37]

Chatty programs are tedious and it is an open question as to how much a program should protect users from their own stupidity.[38] What might (I have my doubts) be appropriate in a single-user situation (or perhaps the near ideal conditions of Bell Labs) is totally inappropriate in a university environment. When it may take from several days to several weeks to get a damaged file restored (if at all), using programs that allow simple destruction of files is perhaps the programming equivalent of riding a motorcycle without a helmet. Similarly, when one wants to concentrate on one's work rather than the peculiarities of programming, it is more than a little insulting to be told that an informative statement of what a program expects is for the "unsophisticated".[39]

As far as possible, **bibliofile** programs check file permissions **before** processing begins, as much to save false runs as to prevent file destruction. Several **UNIX** utilities that do not do this will run, often at some expense, and then abort at the final stage due to inappropriate permissions. Not all files can be tested in advance, but those that can be are. A separate command is required to overwrite an existing file or it must be removed before any output is generated.

bibliofile diagnostics echo the offending command in order to show "metacharacters" that may have been changed by the "shell" and to show spurious characters introduced by keyboard bounce. Partly because of the nested and recursive possibilities of the **UNIX** shell, program names are included in both diagnostics and prompts, but there is a more mundane reason for including the name of the program in the prompt. Like many of those using these programs, I work in an environment subject to constant interruption by students, colleagues, etc. Since logout and login are time-consuming, I usually stay logged in during short interruptions but often forget what I was doing when interrupted. Having a program prompt with its name, as in

(ked)

or

(kord)

at least reminds me where I was.

Not all users might need the named prompt, but plain English diagnostics are useful to all. There is no excuse in any computer system, especially **UNIX**, for anything else. At the very least it is possible to have a table of standard diagostics identified by a number. When an error is encountered, the program in trouble calls a pattern-finding routine (**grep** in **UNIX**) that finds the number in the table of diagnostics and lists the associated message. There should be no market for guides (especially priced at $25 for 24 pages of text) to explain messages from programs.[40] Indeed, there is no good reason for not building good diagnostics into a program itself. Even for the most complex programs, a full set of terse, plain English diagnostics is unlikely to require more than 1 or 2 k of core, and with a machine capable of addressing 128 k, these can easily be included in the source code itself. Even the very explicit messages in **kform**, complete with an arrow pointing to the probable mistake, are very simple to generate as the command-parsing process itself inherently supplies an indication of where a problem occurs.[41]

bibliofile programs do not, however, continually inform the user of what is going on. Conventional terminal screens are small enough

without having one or more lines taken up by status information. Programs that give a blow-by-blow description of what is happening may well distract the user or obscure important information in a stream of trivia. **kord** is the only program that always provides an execution report, and it does so because sorting very large files under heavy load conditions may require so much time that the user becomes frustrated and, thinking something is wrong, sends an interrupt or break. Even in this case the information given is no more than that needed to tell the user the program is indeed alive and well. The sequence is "<indexing>," "<sorting>," and "<rebuilding>".

User Friendliness

In trying to make **bibliofile** into a system of general utility, the author has studiously avoided one approach that is enjoying something of a fad: menus and windows.[42] Aside from the complexity they add to programs, the greatest objection that can be raised against the use of menus and windows is the time their display requires. Over a three-hundred-baud telephone line, it takes a minimum of 65 seconds to draw a full screen, and more if an elaborate display with font changes is used.[43] Moreover, this minimum is achieved only if the menu is an integral part of the program. Then, one must read through the menu and make a selection. In many menu-driven systems, one menu leads to another. It is very easy to spend more time waiting for and reading menus than actually working. Even with faster data lines, menus soon grow tedious, especially if, as with many programs, you cannot shut off or interrupt the menu.

In lieu of menus, **bibliofile** offers command summaries on request, on-line documentation, and extensive error checking and feedback. At the shell level, the user can get a summary of commands for any **bibliofile** program by typing:

prog hint

A printed copy of this "hint," which is organized like a "quick reference card," can be obtained by:

prog nint |lpr -N

From within a program the same "hint" is called by typing a single "?" (question) mark. Fig. 11 is an example of one such "hint" taken from **krect**, a **bibliofile** program used to correct spelling errors.[44]

Figure 11.

```
----------
krect [file names]
----------
commands
----------
?           list commands
r/string    replace displayed word by string (ksed)
s/string    replace displayed word by string (ex)
x           erase the previous command
u           "
c           context search on word
p           context search on word as pattern
l           context search on word
            (show non-printing)
#           mark word as ###word###
            (for subsequent editing)
go          begin processing text
q           quit without further processing
!cmd        execute cmd
----------
```

In most programs these command summaries, contained in a separate routine which is actually compiled into each program, provide nearly instantaneous response to a request for help, subject only to the limits of the transmission rate.[45] Errors also cause the "hints" to be displayed.

The "hint" function makes it easy to keep at least a portion of the documentation up to date. Since the "hints" are compiled into the programs, only a single extra step is needed to update them whenever the program itself is changed.

The user can also examine the documentation for any program by typing

prog help

and get printed copies of the documentation for one or more programs by using a utility called **ksysdoc**.

In writing the documentation, I have emphasized practical examples and tried to anticipate common errors. Because I actually use these programs in my own work and have close contact with others using them, I can base my examples and anticipated mistakes on real-world experience. With the documentation maintained on-line, new examples, explanations, and "bug notices" can be added quickly.

Examples are generated in several ways. First, I try to provide one example to illustrate each option by itself and in what seem to me to be the most probable combinations. In developing each program, I also include a phase where I deliberately abuse the program, giving it bad commands and absurd parameters. If the bad command or absurd parameter is one that can be caught, the program is modified to include an appropriate diagnostic. If the case is ambiguous—what are absurd parameters to one may not be to another—a warning is included in the documentation about what to expect. Through personal use of the programs, I discover shortcuts or quirks, and if these seem to be of general interest, they are included in the documentation. Feedback from users also contributes to both the diagnostics and documentation. Users are much more adept at discovering bugs than are authors.

In part this approach to documentation is common sense, but it is also in part a reaction to the atrocious documentation that comes with **UNIX**. Most program descriptions provide either no examples or purely trivial ones. Because many of the "metacharacters" used by pattern-finding programs and by system utilities are also special to **nroff** and **troff**, it is not unusual to have key elements missing from the examples because some significant character was stripped by **troff** when the documentation was printed.[46] There is enough start-up cost involved in learning how to use any new program without having to try to second-guess a typesetter that eats text. By printing its documentation through **kroff**, which uses fewer "metacharacters" and has good provision for getting around the few it does use, **bibliofile** avoids this problem.[47]

Efficiency

All programs in the **bibliofile** system are written in **C**.[48] Inasmuch as the **UNIX** system itself is written almost entirely in this language, it is automatically the choice for serious work under **UNIX**, for it allows direct entry into the operating system. **C** is a medium-level structured language similar to PASCAL but without the lacunae that make "standard" PASCAL almost worthless as a serious language.[49] As a structured language (like PASCAL), it forces one to think logically and discourages the "spaghetti code" so common to FORTRAN and BASIC.[50]

C provides many features useful for serious programming, including recursive subroutines, user-definable data types, a full set of bit-manipulation operators, and structures (devices) for grouping variables in sets and subsets. Moreover, **C** produces very efficient and *relatively* portable code.[51] On the basis of many years of experience in programming FORTRAN and COBOL, I would suggest that **C** is the most versatile language available for serious software production.

In arguing that learning efficiency is at least as important as execution efficiency, I do not mean to imply that no attention has been given to execution speed. Where **C** offers several alternative routes to the same end, the more efficient construct has been used, *provided the result is readable*.[52] Overall programming structure has been checked for efficiency by use of the **UNIX** utility **prof** (profile), which provides statistics on the number of times each routine is called in a program and how much of the cumulative time is associated with each routine. **prof** tells which routines need improvement and which are worth effort. A horribly inefficient routine that is called only once and contributes only a few percent to overall execution cost is **not** worth rewriting. Running **prof** on **bibliofile** programs has shown that sixty or seventy percent of all time is accounted for by low-level system operations (not subject to programmer control) with the remainder distributed widely over a large

number of my routines. Given programs of the type described here, this is as one would expect.

One case that **prof** did point out was in **kroff**. Originally, **kroff** called a routine that checked for vowels followed by macrons, which were converted into overstruck vowels (To^kyo^ becomes Tôkyô).[53] Rewriting this function into in-line code took ten percent off the running time of the program by eliminating the overhead involved in calling a subroutine for each and every character. Cases of such clearly bad and easily correctable structural errors have, however, been very rare.

This is not to say that I write ultra-efficient code. Presumably a professional programmer could make changes that would raise efficiency. Nevertheless, since the portion of program time not accounted for by system input-output is relatively small, even a one hundred percent improvement in all other routines would result in improvements of only twenty-five percent or less overall. The price for greater efficiency would be code that is cryptic and harder to understand, particularly at a later date, and that involves less error checking.[54] This is a price usually not worth paying.

Conclusion

bibliofile is hardly the "perfect" data base or filing system. There are many aspects that could be improved. A "screen" rather than a "line" editor would add much to the convenience of the system.[55] There ought to be some provision for editing patterns in programs such as **krep** and **kord**. For those who really want something akin to an airline reservation system, there should be some sort of efficient indexing or hashing scheme. Nevertheless, **bibliofile** can, I think, hold its own against commercial programs, even if one does not consider cost.[56]

NOTES

1. Bell has essentially made **UNIX** available for the cost of the tapes containing the system. See David and Susan Fiedler, "Selecting a Small **UNIX** System," *UNIX Review* 1:1 (June-July, 1983), p. 36.

2. Strictly speaking, **dbminit** is a set of very difficult to use routines for managing a hashing-type data base. Not all distributions of **UNIX** even have **dbminit**.

3. These include:
James Murphy, Rhetoric, An Encyclopedia of the History of Rhetoric
Lynn Roller, Classics, Catalogue of Pottery Marks from Gordion
Susan Shimanoff, Rhetoric, A Dictionary of Terms in Discourse Analysis
Special Collections Library, The Hal Higgins Agricultural Collection
Library Associates, Bibliography of UC Wine Publications, 1868-1968
Michael Motley, Rhetoric, Study of Prejudicial Language
Mortimer Schwarz, Law, Catalogue of Presidential Action Committees
Kevin Roddy, Medieval Studies, Bibliography of Medieval Culture
Laura Martinez, Graduate Division, Census of Graduate Employment
Keith Young, Research Division, Faculty Profiles
Terry Weidner, Medicine, Specimen Collection Inventory
Eva Carroad, Primate Center, Secondary Bibliography
Ken Firestien, Library, Checklist of Appropriate Technology
Linda Bickham, Library, Serial Articles in Biological Science.

4. This might be more appropriately called a "tags" file to distinguish it from the notion of "keys" for indexing content.

5. Users are free to extend, modify, or replace this ".keys" to given conformity with MARC or other "standards."

6. **ked** defaults to a 2048 buffer. This can be raised by a "$buffer=nnnn" statement (in the ".keys"). If a given machine has enough core, the maximum number of cards addressable by the **ked** may be raised by a "$maxcards=nnnn" statement.

7. This aspect of **UNIX** is discussed in Brian W. Kernighan and Rob Pike, *The UNIX Programming Environment* (Prentice-Hall, 1984), p. 44. This simple file structure is one of the most important aspects of the **UNIX** system. In effect, **UNIX** was an "integrated" system a decade before "integration" became a buzz word in small system software.

8. The clarity of **ked** records is in stark contrast to those of most commercial or institutional data systems. For examples of such schemes, see H. S. Heaps, *Information Retrieval: Computational and Theoretical Aspects* (Academic Press, 1978), chap. 3.

9. For examples of the problems caused by more complex coding schemes, see Kristina M. Brooks, "The Online Transfer of A Pandora's Box," *Database* (February, 1982), pp. 18-21.

10. Some line-printer and terminal software does this automatically but in arbitrary fashion. A simple **bibliofile** utility **kfold** does intellligent line folding on the first blank after n (default 64) characters.

11. Very long lines are unacceptable to some utilities. Again, **kfold** will make the records acceptable.

12. At UCD the bankruptcy of an account usually requires moving files to an account with money. If the files were indexed with specific disk addresses, address tables would have to be rebuilt every time a file was moved.

13. Fixed format systems are derived from the 80 column "IBM" or Hollerith card. This was in turn derived from the control cards of the Jacquard loom invented in 1745. In this sense, any data base system using fixed length records may be regarded as using eighteenth century technology.

14. An alternative approach might use a byte-stream with numeric tags (keys) both marking off fields and supplying an indication of what follows. This could be done by using one eight-bit byte with the high bit on to distinguish it from the ascii character set (which does not use the high bit). While this would limit the system to 127 data fields, it would produce some savings on overall file size and speed of input-output.

Assuming a file with an average of ten fields per record each averaging twenty characters, the savings with a numeric scheme would be 67 percent on keys (tags) relative to a two character (plus one separator) alphanumeric system. Because the keys (tags) are only a small percentage of each record, a 67 percent savings there translates into only a 16 percent savings on each record. Further-

more, this savings declines as the average field size increases becoming 13 percent for 40 character records and 9 percent for 80 character records.

Except for linear (sequential) processing, the space savings of a numerical scheme does not translate into equivalent time savings. Random access reading and writing has a sunk cost for each seek that is independent of the number of characters transferred.

The small gain in efficiency coming from numeric coding requires giving up the self-documenting character of **bibliofile** records. It can produce bizarre output with simple dump routines because the numeric codes are non-ascii. High bit numeric coding also requires a translation filter for use with other **UNIX** utilities.

Similarly, storing numerical data in binary form would sometimes produce a savings in file size. (12345 requires 5 bytes in ascii, two in binary.) But, translation would be required whenever the file was to be handled by **UNIX** utilities.

15. By "bleeding" I mean the case in a fixed-length system where excessive length entries are not detected and the excess from one field "bleeds" into the next. The popular micro-computer program **dBASE** is a good example of bleeding.

16. Pattern finding (**krep**) does not require reference to the keys as such.

17. Many of those using these programs at UCD begin by using public access terminals. These are used and abused by students from bonehead FORTRAN classes and by game players.

If required, leading and trailing blanks can be forced through by the standard **UNIX** convention of quoting with the escape (/) character.

18. Some operating systems use the term "chaining" to describe this type of mechanism.

19. Even with the **UNIX** system, one has this problem. For example the manual for **join** notes that the conventions for the overlapping programs **join, sort,** and **uniq,** are "wildly incongruous." "join," *UCD UNIX Programmer's Manual* (1982).

20. Due to the need to buffer output for efficiency, it is possible to lose the very last record edited or entered.

21. **ex** and **vi** developed at UCB have these features and more. **ked** was, in a loose sense, patterned after **ex**, but the code is the author's own plus some inspiration from Brian W. Kernighan, *Software Tools in PASCAL* (Addison-Wesley Publishing Company, 1981), chap. 6.

I have found that even some people holding the title of "systems programmer" do not know how to implement an "undo" function or think it too difficult to "reverse" a command. No reversing is done. One merely keeps two alternating copies of the buffer's address table, updating this *after* each change. Although uncommon in editors, this extremely useful function involves only a few lines of code.

22. Without specification **ked** supplies address tables for 2048 records, each 2048 characters (4.2 million characters total). Some users have doubled both parameters for a directly addressable file of 16.8 million characters.

23. In practice the actual file will not be as large as simple multiplication of these factors would imply. To edit a record rather than simply enter it, there must be a free area equal to the longest single line in the record. Thus if the card buffer is 2048 and the longest single line in any record is 512 charcters, the effective usable record size is 2048 - 512=1536 and the effective addressable file size is 1536 * 2048 = 3.1 million characters. This is still far more than the usual **UNIX** editors can handle.

The failure of **UNIX** utilities to make effective use of other attractive features of the operating system is discussed below under "interaction."

Other programs in the system are, for all practical purposes, totally open ended.

24. Readers unfamiliar with **UNIX** may be confused by its capability to use whole programs (**sort** in this case) much as one would use functions from within a higher level language. **sort** is in fact an independent program that is initiated by **kord** which also collects its ouput.

This aspect of the **UNIX** system is discussed in S. R. Bourne, *The UNIX System* (Addison-Wesley, 1983), chap. 4 and in Kernighan and Pike, *UNIX Programming Environment,* chaps. 3, 4, and 5.

25. The program also has miserable documentation, but that is true of most **UNIX** programs.

A program similar to the **UNIX sort** is described in Kernighan, *Software,* chap. 4

26. See James Martin, *Computer Data-Base Organization* (Prentice Hall, Inc., 1975), p. 433.

27. **sed** (the **UNIX** stream editor) would be more efficient but **ex** or **ed** will overwrite its input saving the novice from worrying about the creation and deletion of a scratch file.

28. There are a wide variety of scripts that would do the same thing. The example uses an attribute of the "Bourne" shell called a "here" text. For a description of these, see S.R. Bourne, *An Introduction to the UNIX Shell* (Bell Laboratories, 1978), pp. 7-8. The actual editor script is a straightforward substitution, familiar to even the most novice user of **UNIX**. The role of the "%" is explained below.

29. Following the fairly standard practice of **UNIX** software, I have used '%' as a token to stand for the current file name. This '%' is not the same as the '%' of the script. There the '%' symbol was used because it was very unlikely that this would appear in any book or article title in the humanities!

30. Most of the items in this bibliograpy are from Kate L. Turabian, *A Manual for Writers of Term Papers, Theses, and Dissertations.* (University of Chicago, 1973). It is used primarily to test style sheets.

31. These figures were generated by **kstix,** a utility intended to give an indication of appropriate buffer sizes and costs of storage. The cost has been omitted here because it reflects the billing peculiar to UCD.

32. The **bibliofile** record structure does not preclude a more complex and possibly faster indexing or relational scheme.

33. For a description of these two utilities see Lee E. McMahon, *SED - a Non-interactive Text Editor* (Bell Laboratories, 1978) and Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger, *Awk - A Pattern Scanning and Processing Language* (Bell Laboratories, 1978).

34. Here again, the basic principle is to build on conventions the user will already be familiar with or will have to learn if he or she wants to use the **UNIX** system beyond the most elementary level. I frankly consider the **nroff** and **troff** instruction language to be among the worst I have encountered in a decade of programming. The dilemma presented is either to start from scratch and develop a language I consider appropriate thereby giving the user an entirely new set of trivia to contend with or to build on something the regular user of **UNIX** already knows or will have to know in the future and try to make it a bit less painful. I have opted for the latter approach. For examples of the **nroff**and **troff** instruction language, see Brian W. Kernighan, *A TROFF Tutorial* (Bell Laboratories, 1980).

35. UC Berkeley charges only for connect time. A cpu-intensive program such as **awk** is economically tolerable in this case, though even with "free cpu time," the overall slowness (real time) of the program makes **kform** that much more attractive. UCD charges for both cpu and connect time.

awk also has several bugs that show up frequently in the applications to which **bibliofile** is applied. The formatted print statement mishandles strings containing '%' and during long runs **awk** bombs and dumps core for no apparent reason. This is probably due to an error in the memory allocation scheme which gradually allows "garbage" (to use LISP terminology) to accumulate until the program runs out of core.

36. Some distributions of **UNIX** still contain this version of **ed!**

37. The **UNIX** utility **expr** has my vote as the worst in the system, although I could make a strong case for **sort** and **find** as well.

38. See Kernighan, *Software,* p. 191.

39. Eric Shienbrood, "more," *UCD UNIX Progammer's Manual* (1982). This program is part of the 2-BSD **UNIX** system from Berkeley.

40. Henry McGilton and Rachel Morgan, *Responses from UNIX Commands (International Technical Seminars).*

41. **kform** uses "recursive descent," a common compiler technique, to generate its code. All the variables required by the error-pointer routine are generated as part of the parsing process. The error-pointer routine itself is about twenty lines of code, much of which is concerned with handling terminals that do not automatically fold along lines.

42. In some extreme cases these are combined with "mice" (a kind of pointer) and "icons" (graphic representations of menu items). Both of these are more marketing gimmicks rather than serious aids to computing. Aside from questions about reliability of the mice, their use presumes a clean desk top and requires removing your hands from the keyboard. The icons (a "trash can" indicates file removal, a "scratch pad" indicates a scratch file, etc. in Apple's **Lisa**) are simply absurd where not actually insulting. For a description of **Lisa,** see Gregg Williams, "The Lisa Computer System," *Byte*

8:4 (April, 1983), pp. 33-50. For doubts about the garbage and vermin approach to user interaction, see "Letters," *Byte* 8:2 (February, 1983), pp. 16-27.

It would be comforting to be able to say that Apple's approach represents something of an extreme. Unfortunately, it does not. **WHATSIT**, a data base system for microcomputers, engages in "pidgin English" dialogues with users. (E. G. Brooner, *Microcomputer Data-Base Management* (Howard W. Sams & Co., 1982), p. 69. One can only hope that as programming matures as a discipline, such approaches will die natural and deserved deaths.

43. Another consideration is that **bibliofile** programs must work with **several dozen** different terminal types.

44. **krect** uses the output from the **UNIX** utility **spell** to make global changes on **nroff/troff** and **bibliofile** files.

45. Having the "hints" in a separate file would slow response time. Even the most verbose "hints" take up no more than 1 or 2 k of core.

46. Examples include **lorder** and **tar** in *UCD UNIX Programmer's Manual* (1982). This also appears to be the problem with some of the examples in the documentation for **M4**. See Brian W. Kernighan and Dennis M. Ritchie, *The M4 Macro Processor* (Bell Laboratories). I do not know if this is peculiar to the local editions or not.

47. **kroff** is the **bibliofile** equivalent to **nroff** and **kroff.** It accepts a substantial subset of **nroff** commands but runs five to eight times faster than **nroff** while putting a much smaller demand on system resources. It can also be operated interactively to preview text and tables. This makes it substantially more convenient than the "batch style" **nroff.**

48. The standard work on **C** is Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language* (Prentice-Hall, 1978). See also the special issue "The C Language," *Byte* 8:8 (August, 1983), pp. 46-285. Individual articles in this issue provide additional support for many of the "philosophical" points I have tried to make here.

49. Standard PASCAL has several major flaws such as no method for early exit from a loop and an unbelievably tedious mechanism for array initialization. Some of these problems are noted in Kernighan, *Software,* pp. 27-29.

50. **C** does have a "goto" that can be used for "rat's nest" programming. There are no goto statements in **bibliofile** programs.

51. The problems that have appeared in moving **bibliofile** programs to other systems, principally those at the University of California, Berkeley, include:

(a) **bibliofile** expects the "Bourne shell" and some system calls must be changed for other shells. These calls are defined in a single header file to make it simple to change all programs at once.
(b) Some programs such as **kord** and **kroff** will have lower record-size limits if they are compiled on machines (other than the PDP 11/70 series) that do not support independent data and instruction areas. On Digital Equipment Corporation VAX machines, the core-limit essentially disappears.
(c) Some **C** compilers may have trouble with my mixed use of pointers and integers. The **UNIX** utility **lint** will identify these cases which are easily correctable by installing "casts" in the source code.

Compatibility considerations are discussed in Kernighan and Ritchie, **C**, pp. 211-213 and in S. C. Johnson, *Lint, a C Program Checker* (Bell Laboratories).

52. For example, **C** allows both FORTRAN-style array indexing and "indirection." The two examples below both copy an array:

```
for (i=0;b[i] != NULL; i++) a[i] = b[i];
while (*b != NULL) *a++ = *b++;
```

The second is substantially more efficient than the first and not notably harder to read.

53. **nroff** requires T *^oky *^o to accomplish the same thing.

54. Slower speed for my routines results primarily from two sources. First, I use more discrete routines to improve readability. There is more overhead in calling a routine than in having in-line code as the **kroff** example demonstrates. Second, I have included explicit boundary checking on arrays to prevent silent truncation or contamination of data from unexpected input.

55. The UCD Computer Center has developed a data entry program (**dbenter**) that works with **bibliofile** records and provides screen editing. Due to bugs in **curses**, the UC-Berkeley terminal driving package, this program cannot work with records that require more than one terminal image to display. It also lacks the recovery and undo provisions of **ked**.

56. **bibliofile** source codes, PDP 11/70 executable elements, and documentation are available to **non-commercial** users on a **zero cost zero support** basis. (The same as for **UNIX** itself!) To get a copy, submit a standard 2400 ft. tape to Earl H. Kinmonth, History Department, University of California, Davis, Davis, California, 95616. Tapes are normally supplied in **UNIX** "tar" format. Turn around is a function of the academic year and other demands on my time.