depth of 3. Anyone who thinks he needs more depth is invited to consult a psychiatrist.

APPENDIX A

CONVERSION

A few words about copying files and converting formats.

1. To copy files from disk to disk, disk to dectape, or dectape to disk, use the system COPY command.

2. To convert SOS files to TECO files, use 'E,S' or <SUBSYS>PIPPY. Either method removes line-numbers and null padding from the file, but leaves formfeeds in. E,S will delete the intermediate SOS copy of the file, iff the strip is successfully completed; otherwise, it remains the current file.

3. To convert TECO files or other files without line numbers to SOS files, simply read them with SOS. SOS will number them by 100 and insert a page mark if there are more then 999 lines on a page. Any form feed which is the first character on a line will be converted into a page mark. "Bare" <return>'s are deleted and "bare" <line feed>'s are changed to <return> <line feed>. An Ascii 37 in the file is presumed to be a TENEX end-of-line signal, and is converted without comment into <return><linefeed>.

4. For those users sending files to UCLA's 360-91, a program here is available which converts SOS files to respectively-numbered card-image files. The program is <CCN-KAY>A2CWP.SAV "Ascii-to-Card with Pagemarks", and converts SOS numbering as follows:

Columns 73-75 == the SOS page, and
76-80 == the SOS line.

If the text of the source SOS-line is > 72 characters in length, the line is split at char 72 onto a second line, with card-columns 73-80 having asterisks.

There is another program, <CCN-KAY>UPDCOM "update-compare", which is a modified srccom. It takes two SOS files and generates a new file with just the updates that were made, plus embedded insert-delete-replace-number commands ... this file is then sent to UCLA rather than sending the complete source each time. A program at UCLA takes the commands and generates an updated source there, as shown in <HEARN>JCL.

APPENDIX B

SUMMARY OF ERROR MESSAGES

Fatal:

DDE	DEVICE OUTPUT ERROR
DIE	DEVICE INPUT ERROR
DNA	DISK NOT AVAILABLE
FNF	FILE NOT FOUND
ICN	INTERNAL CONFUSION
TNX	TENEX CONFUSION
ILUUO	ILLEGAL UUO
NEC	INSUFFICIENT CORE AVAILABLE

Non-fatal:

ILC	ILLEGAL COMMAND
ILFMT	ILLEGAL LINE FORMAT
ILR	ILLEGAL REPLACEMENT
IRS	ILLEGAL REPLACEMENT STRING
ISS	ILLEGAL SEARCH STRING
ITD	ILLEGAL TRANSFER DESTINATION
LTL	LINE TOO LONG
NLN	NO SUCH LINE(S)
NNN	NO NEXT LINE
NSG	NO STRING GIVEN
NSP	NO SUCH PAGE
ORDER	OUT OF ORDER
STC	STRING TOO COMPLEX
STL	STRING TOO LONG
UNA	DEVICE NOT AVAILABLE
TMS	TOO MANY STRINGS
WAR	WRAP AROUND
BKO	BREAK OUT
XRO	NOT DURING READ-ONLY

APPENDIX C

SUMMARY OF COMMANDS

The following is a brief summary of SOS commands. Those arguments enclosed in () may be omitted. Where several arguments appear, seperated by |, it means that any one of these (but only one) may be used. [] are used for grouping.

```
Alter   A<range>
           <space>  next character
          <rubout>  last character -- also ^A
                 C   Change
                 D   Delete
                 I   Insert
          <return>  end alter mode
                 Q   Quit              -- also ^X
       <control U>  start over
                 S   Skip
                 K   Kill
                 R   Replace
                 L   Line
                 P   Print             -- also ^R
                 J   Justify

Beginning
        B

Break-out
        <control B>

Comment
        ;

Copy    C<dest>(_<file>(,S)),<source>(,<incl>)(,<inc2>)

Delete  D<range>

End     E (<file name>)(,S)

Find    F((<string>)<altmode>(<range>)(,A|,N)(,E)(,<number>))

Go      G (<file name>)

Help    H
Insert  I<line>(,<increment>)

Join    J<line>

Justify Center
        JC<range>
```

```
Justify Left
        JL<range>


Justify Right
        JR<range>

JUstify
        JU<range>

List    L(<range>)(,S)

Mark    M<line>

Number  N<increment>,<range>(,<starting number>)

display extend
        Q<range>

Print   P(<range>)(,S)

Replace R<range>(,<increment>)

Substitute
        S((<ostrng><altmd><nstrng>)<altmd>(<rng>)(,D|,N)(,E)(,<number>))

Transfer
        T<dest>,<source>(,<incl>)(,<inc2>)

save World
        W(<file name>)

eXtend  X<range>

display alter
        Z<range>

give information
        = =|_|FILE
        = .|BIG|CASE|INC|ERROR|STRING|PMAR|LMAR|RMAR|MAXLN|SAVE|ISAVE

set
        _UPPER|LOWER
        _DPY|M33|M37
        _C128|C64
        _NOVICE|EXPERT
        _SHOW|SUPPRESS
        _[PMAR|LMAR|RMAR|MAXLN]=<number>
        _INC=<number>
        _[SAVE|ISAVE]=<number>
```

APPENDIX D
STANDARD TEXT FORMAT

The format of a standard text file is defined to be as follows:

A line number is a string of 5 ascii digits, left justified in a single word, and having bit 35 a 1. A line consists of a line number followed by the text of the line in ascii characters, left justified in words with bit 35 a Ø. The first character of the text is a tab and the last two characters are <return> and <line feed>. The characters null (Ø) and delete (177) may not appear in the text. The last word of the line is filled with nulls to make a complete word if necessary.

A page mark is a word containing 5 ascii spaces (4Ø), left justified in a word with bit 35 a 1, followed by a word containing the characters <return>, <return>, <form feed>, <null>, <null> and with bit 35 a Ø.

Lines are placed into records starting with the first word. A line is never broken across 1Ø/5Ø block boundaries. No Ø words appear in a record except after the last line of that record. All unused words in a record are Ø.

SPELL

SPELL is a program that checks text files for correctness of word spelling. In addition to the spelling check, SPELL provides a facility for correcting words that it thinks are misspelled. The program was originally written by Ralph E. Gorin at Stanford University. It has been adapted to TENEX and TOPS-20 and considerably augmented at BBN.

SPELL reads through the input text file, checking each word against its dictionary. All words that it finds in the dictionary or that it can find after stripping off common prefixes and suffixes are copied through to an output file. When it comes upon a word it can't recognize, it tries to guess a correction and then asks you how to treat this word. You can have SPELL accept the word as correct, add the word to its dictionary, or correct the word in several ways. In this way, the output file becomes a corrected version of the input file.

NOTE: SPELL has no knowledge of syntax, so it will readily accept things like "JOHn red the book", even though "red" is incorrect usage. SPELL can only check each word against its dictionary to see if it corresponds to any correct spelling. Note also that SPELL takes no special note of "JOHn", since it does not consider case to be significant. You shouldn't come to believe that SPELL will catch all your typographical and spelling errors; it will detect many errors that a human reader could easily pass over, but it is no substitute for proofreading!

SPELL's internal dictionary contains over 40,000 words (including RUNOFF and MRUNOFF commands). Because SPELL's built-in dictionary cannot possibly contain all the words and proper nouns in an individual user's vocabulary, SPELL contains provision for the user to maintain a private dictionary file (or files) to augment SPELL's built-in vocabulary. For all but casual users this practice is highly recommended.

NORMAL USE OF SPELL

SPELL's commands and switches are all single letters. A few commands take an optional dictionary number following the letter. All typeins to SPELL must be terminated by carriage return. The editing of typed input can be done in the usual way, using the conventions of either TENEX or TOPS-20.

The editing characters are:

Char-delete	^A, Rubout, or ^H
Word-delete	^W
Line-delete	^Q, ^U, or ^X
Retype line	^R
Quote character	^V

(1) When SPELL is started, the first thing it does is to see if you have a default private dictionary file; if there is a file in your login-directory named DICT.SPELL, this is assumed to be a dictionary file and loaded as an "incremental" addition to SPELL's built-in dictionary. (When you subsequently exit from SPELL with the "E" command (see paragraph (5) below), SPELL will automatically update this file if necessary.) Proceed to paragraph (2) below.

If no DICT.SPELL file was found, SPELL will ask if you want to load a private dictionary file by asking:

Do you want to augment the dictionary?

If you do not want to load a dictionary file (or have none to load), type "N" (or just <cr>) and skip to paragraph (2) below. (If you forget to load a private dictionary at this point, you'll have other chances later on.) If you answer "Y", SPELL will type:

Dictionary file name:

After you type in the name of the dictionary file, SPELL will type:

Type "I" to mark these as incremental insertions:

The usual practice at this step is to type the "I", for that means that the new words will be marked as "incremental" additions, so that they would be included in an incremental copy of the dictionary at the end of the session (if you make one - see (5) below); not typing "I" means that the new words become merged with SPELL's internal dictionary.

After the dictionary file is loaded, SPELL will again ask if you want to augment the dictionary, thus permitting more than one dictionary file to be loaded. Eventually you'll give a negative answer and thus go on.

(2) SPELL will next ask if you want to set any optional mode switches:

```
Mode switches (zero or more of P,U,M,N,A,T,Q, or ?):
```

The usual response is just a carriage return to specify no switches. The other options are:

P Pickup mode. You will be asked to specify a page and line number at which to "pick up" the spelling checking. When you have a partially corrected file, this mode enables you to skip over the initial portion that has already been corrected. The input file is copied to the output file without checking until the page and line specified, at which point spelling checking begins.

U Upper case mode. In this mode, each new word that is entirely upper case will not cause SPELL to ask you about it, but will be inserted in a special dictionary. You will be asked to specify a dictionary number for the upper case words (see HOW TO USE MULTIPLE DICTIONARIES below). This mode can be useful if your file contains many special "words" that are written in upper case, such as logic symbols, opcodes, or program variable names, and you don't want SPELL to query you about each and every one. Note, however, that SPELL does NOT check to insure that subsequent occurrences of these words are also entirely upper case.

M Misspellings. Put misspellings (and their corrections) in a special dictionary. You will be asked to specify the dictionary number. Normally, when you correct a misspelling found by SPELL, the misspelling-correction pair is put in SPELL's main dictionary so that subsequent occurrences can be automatically corrected, but these misspelling-corrections can't be saved for future use. This mode lets you specify a dictionary for these misspelling-corrections; this dictionary can be the default incremental dictionary (#1) or a special one. This mode can be useful if you misspell more frequently than you mistype, and you'd like to accumulate instances of the words you misspell for use in checking other text files.

N No suffix-stripping.

A No prefix-stripping. The suffix- and prefix-stripping algorithms are useful and clever, but far from foolproof. You can use the N and A switches if you prefer to use SPELL without them.

T Training mode. SPELL will treat the input file as a training set rather than a file to be corrected. See the discussion of TRAINING MODES below.

Q Q-training mode. Similar to T: see the discussion of TRAINING MODES below.

(3) You will then be asked to specify the names of the file to be checked, the corrected output file, and an exception file:

```
Name of the file to check and correct:
File name for corrected output:
File for exceptions:
```

If you specify the input file as INFILE.EXT, then typing <esc> to the "corrected output" prompt defaults the output file name to INFILE.EXT; similarly, typing <esc> to the "exceptions" prompt defaults the exception file name to INFILE.EXCEPTIONS. (See THE EXCEPTION FILE below for a description of that.) If you type <cr> to either of those prompts, then the respective file will not be created at all.

(4) After you have specified all the files, SPELL will respond with "Working..." and start checking the input file for spelling errors. While it is doing so, there are three kinds of occurrences that will cause SPELL to type out on your terminal: (a) unknown words that SPELL needs to ask you about, (b) known misspellings, and (c) words that SPELL has matched by means of affix-stripping rules. Only the first of these requires you to type anything in reply.

(4a) Each time SPELL encounters a word that it doesn't recognize, it will ask you about it by typing the page number, the line number, and the line in which the word occurs, followed by the word itself, and whether it has any guesses for correcting it. (Note that the page number typed by SPELL refers to the number of page-feeds in the input file; it generally won't correspond to the page number of the finished document.) If SPELL has one to three guesses, it will type them out also. For example:

```
Page 1:34
reads through the imput text file, checking each
IMPUT  I guess: (1) INPUT or (2) IMPUTE
*
```

The asterisk prompts you to tell SPELL what to do with the excepted word. At this point, you have a choice of several commands, some of which (S, C) are dependent on the number of guesses. (Typing "?" will get you a summary of the allowable commands.)

A Accept this word, this one time.

I Insert. Accept this word and insert it in the dictionary so that subsequent occurrences of this word will be recognized and accepted. Words that are inserted this way are marked as incremental insertions, and may be copied out to form an auxiliary dictionary file.

C(n) Correct this word with SPELL's (n-th) guess. If SPELL has volunteered just one guess, then type "C" to use it as the correction. If SPELL has shown 2 or 3 guesses, or if you have typed out many guesses with the "S" command, then type "Cn" to use SPELL's n-th guess as the correction.

S Show all of SPELL's guesses. If SPELL has more than 3 guesses for the unknown word, it won't type them out unless you request it with this command. After SPELL has done this, you may select one of the guesses with the "Cn" command described above.

R Replace this word. SPELL will ask you to type the replacement word. If the replacement word is not already in the dictionary, SPELL will give you an opportunity to insert it.

If the misspelling is due to an omitted space between words, use the "R" command to retype the words with the space.

L Load a private dictionary file to augment SPELL's built-in dictionary, then reconsider this word. You will be asked for the dictionary file name.

X Accept this word and finish. The word will be accepted. Then the remainder of the input file will be copied without checking to the output file. This is useful if you are only part way through a file and you wish to stop without losing the corrections already made. (The next time you use SPELL, you can use "Pickup" mode to resume checking this file at the same page and line number.)

If the X is followed by a number n, it means something slightly different. It means to suspend spelling checking for the next n lines (including the current one). This can be

useful for skipping a portion of text containing nonword character strings.

W Save the incremental insertions. After you type "W", you will be asked for a file name. Then an incremental copy of the dictionary will be written into the file. After the copying is complete, you may decide what to do with the excepted word.

D Display the line and offending word again. The line that is displayed will not have any corrections shown in it.

(4b) SPELL knows about many common misspellings, and as you process the input file, it also remembers corrections you make to misspellings it finds. If SPELL encounters a word that it already knows is wrong, it automatically performs the correction, and it informs you of this by typing (for example):

```
Page 1:179
as seperate but equal branches of government.
seperate ==> separate
```

(4c) If SPELL cannot find a word in its dictionary, it applies various prefix- and suffix-stripping rules, to see if it can recognize the stem after these affixes have been removed. For each such word it finds, SPELL will:

- type it on your terminal, as for example:
  UNSTEADINESS = UN+STEADY-Y+I+NESS
- note it in the exception file, and
- enter it into the dictionary (specifically dictionary #31)

Since each affix-match is entered in the dictionary, if the same word is encountered again, it will be accepted with no further action. Thus such an affix-match type-out signifies only that the file contains at least one occurrence of it. Although the affix-stripping rules are quite effective, they are far from foolproof (e.g., CHOSES would be accepted as CHOSE+S). Therefore SPELL types out these affix-matches so that you can monitor their validity.

(5) When the input file is exhausted, all files are closed, SPELL types "Finished processing the input file." and enters an exit-command sequence:

Type E,I,N,C,A, or ?:

Again, you have several choices:

E Exit from SPELL to the EXEC. If SPELL had automatically loaded a DICT.SPELL dictionary file at startup and words were incrementally added to that dictionary, an updated DICT.SPELL file is automatically written.

I Incremental copy. Make an incremental copy of the dictionary. All words that you inserted while running SPELL (and loaded from private dictionary files) are copied to a file. SPELL asks for the file name. The words in this file will not normally be in alphabetical order, but rather sorted only on the first two letters and the length. (If you type IS (S for "sorted"), the file will be sorted into alphabetical order, but BEWARE! this consumes quite a lot of CPU time.)

N Numbers. Type out statistics about how many words were processed, how many spellings and misspellings are in the dictionaries, and the time used by SPELL.

C Correct. Go back and correct another file.

A Augment the dictionary (load another dictionary file), set new mode switches, and correct another file.

SPELL DICTIONARY FILES

Dictionary files for use with SPELL (and as copied out by SPELL) are ordinary text files, which may be edited and listed. They contain two types of entries: correct spellings and misspelling-correction pairs. The format is one dictionary entry per line. Each entry must be composed of alphabetic characters (apostrophes are permitted inside a word) less than 40 letters long. The entries need not be in alphabetic order. Upper and lower case letters are not distinguished.

Misspelling-correction pairs are put on a single line in the format "misspelling>correction" (e.g., ARGUEMENT>ARGUMENT).

When you load a private dictionary file, any words in the file that are already in SPELL's main dictionary are not duplicated. Hence, if your words are marked as incremental additions, then in a subsequent incremental dictionary copy, these duplicate words will not be copied out.

SPELL comes with a large built-in dictionary, but even a very large dictionary cannot encompass all the jargon words and proper nouns in an individual user's working vocabulary. SPELL lets you make incremental additions to the dictionary as you encounter new words, but it cannot remember them from one session to another. Therefore it is wise for you to maintain a private "incremental" dictionary file in which you save such words after each use and which you load into SPELL each time you use it.

SPELL's "default private dictionary" feature makes this particularly easy to do. When you start up SPELL, if there's a file in your login-directory named DICT.SPELL, it will be automatically loaded (into dictionary #1). When you exit SPELL via the "E" command, if you have added words to the incremental dictionary, then SPELL automatically writes an updated DICT.SPELL file before stopping. If you don't have a private dictionary file to start with, simply create an empty file named DICT.SPELL, or the first time you use SPELL, create a dictionary file named DICT.SPELL with the "I" command before exiting.

Of course, you can name your dictionary file (or files) anything else. In that case, you will have to load and update it yourself, using the commands described above in NORMAL USE OF SPELL.

HOW TO USE MULTIPLE DICTIONARIES

SPELL has a set of features whereby you can cause the creation of several disjoint incremental dictionaries. In this way, you may collect several dictionaries of special vocabulary classes.

The section above called NORMAL USE OF SPELL talked only of SPELL's "main" dictionary and "incremental" additions to it. In fact, SPELL contains 32 dictionaries, numbered Ø to 31. Dictionary Ø is the main dictionary; words can be added to this one only by reading in a private dictionary file at SPELL-startup time. Words that are inserted "incrementally" are marked as being in dictionary 1, unless you specify otherwise. Dictionary 31 is reserved for holding affix-matching words. You may specify a dictionary number at several places during the running of SPELL by appending a dictionary number to some commands; if no number is specified, the dictionary number will default to 1.

The following places are where you can specify which dictionary to add words to:

1. When loading a dictionary file at SPELL startup time, you are asked to "Type "I" to mark these as incremental insertions." Responding with "In" (where n is a decimal integer between 1 and 30) means to add the words to dictionary number n.

2. While the input file is being checked, after SPELL has asked you about an unrecognized word, type "In" to insert that word in dictionary n. If, instead, you load a dictionary file with the "L" command, to have it go into dictionary n, type "In" to the prompt that says "Type "I" to mark ...".

3. After replacing a word (with the R command), if SPELL asks you whether you want to enter the word in dictionary, then type "In" to insert the replacement into dictionary n.

When requesting an incremental copy of a dictionary to a file, you may specify the particular dictionary to be copied (1-31). This is appropriate in two cases:

1. After some word has been asked about, the command "Wn" will cause dictionary number n to be copied.

2. During the exit sequence, the command "In" will cause dictionary number n to be copied. (If you want dictionary number n to be sorted alphabetically, type "InS".)

HINT: In the course of correcting a file, it is likely that you will be asked about words that you wish to have accepted during this file, but which you don't wish to have saved in your incremental dictionary(s). In these cases, simply insert them in a "throwaway" incremental dictionary (such as dictionary 9), which you don't bother to copy to a file when you're finished.

THE EXCEPTION FILE

In addition to the corrected output file, SPELL may produce an "exception file," in which are noted those places in the input file where SPELL encountered a word not found in its dictionary. Each word that SPELL asked you about and each automatic misspelling correction is noted along with the line in which it occurs. Also noted is (the first occurrence of) each word SPELL recognized by dint of stripping off prefixes and suffixes.

The exception file also has a special use in Q-training mode, as described below.

TRAINING MODES

SPELL has two "training" modes, in which it treats the input file as a training set rather than a file to be corrected. All words in the file that are unfamiliar to SPELL are entered in the dictionary as incremental insertions. Afterwards, you may do an incremental copy of the dictionary to a file, examine and edit it to remove misspellings and other inappropriate entries, and subsequently use it as an auxiliary dictionary file. This feature is provided for the purpose of creating specialized dictionaries of jargon or technical words from existing text files.

The training modes are selected by typing "Q" or "T" in response to the "Mode switches" request when SPELL is started. In both cases, you are asked to specify a dictionary number in which the unfamiliar words are entered; if none is typed, 1 is assumed. There is no output file.

T Training mode. Operates as explained above.

Q Q-Training mode. Identical to T, but with this additional feature. If any of the unfamiliar words is "close to" a word that SPELL does know, it is also output in the exception file. In this way, SPELL calls to your attention the fact that these words may be misspellings. The exception file contains only such words.

MISCELLANEOUS FEATURES OF SPELL

A text file may contain portions (such as quoted dialect or program excerpts) on which it is undesirable that SPELL perform its checking and correcting. You can tell SPELL to turn off its spelling checking by including in the file a line containing only ".<any-char>NOSPELL". Spelling checking is turned on again by a line containing ".<any-char>SPELL". (Case is not significant in NOSPELL/SPELL.) Most text formatters permit comment lines of the form ".*Comment" or ".;Comment", so these SPELL-control lines will be transparent to them. For example, in a MRUNOFF file:

```
This part of the text will be checked by SPELL.
.*nospell
Any "misteaks" in this part won't be seen by SPELL because
of the .*nospell comment line above.
.*spell
```

SPELL will read either SOS or TECO format files for the file to be corrected; the output file will be written in the same mode, with the same SOS line numbers, if present, that the original file had. Dictionary files may be either SOS or TECO format.

When a word is corrected, the output file will be rewritten with either upper case, lower case, or mixed (first letter upper, the remainder in lower), depending on the cases of the first two letters in the original word. Note that this will be incorrect in a small number of cases (such as McCarthy); SPELL will type a warning in these cases.

When SPELL updates the default dictionary file (DICT.SPELL), it sets the file retention count to 1. On TOPS-20, this has the effect of deleting all but the current version of that file. (On TENEX, this has no effect.)

Before SPELL asks you about a word it doesn't recognize, it tries several spelling correction heuristics in an attempt to "guess" the correct word, on the assumption that the excepted word is a misspelling and not a novel word. For short words, this will usually result in a large number of guesses, but for longer words, this will frequently result in only one or two guesses. The kinds of errors checked for cover a wide variety of sins of misspelling and typographical error:

1. one wrong letter
2. one missing letter
3. one extra letter
4. two transposed letters
5. an omitted space between words (sometimes)

USE OF SPELL UNDER HERMES

HERMES, BBN's electronic mail handling system, permits SPELL to be used to process text fields. When SPELL is invoked from HERMES, excepted words are handled as explained in paragraph (4) above in NORMAL USE OF SPELL, but the preliminary and final procedures (paragraphs (1-3, 5)) are virtually eliminated.

1. The "default private dictionary" works as described above. You are not specifically asked if you want to augment the dictionary, but the "L" command at excepted-word time gives you an opportunity to do this if desired.

2. No mode switches may be set.

3. When SPELL is finished, the default dictionary file is updated. If no default dictionary file exists, but words have been added to the incremental dictionary (#1), you are asked if you want to save it (create a dictionary file). There is no opportunity to save any other dictionary.

SRCCOM

A Source Compare Program

The program SRCCOM on TENEX is an adaptation of the verify option of UTILITY on the XDS 940 with several improvements and additional features. The name is the same as DEC's source compare program, but for our purposes SRCCOM has been TENEXized and is now 50% faster.

SRCCOM provides a facility for comparing two symbolic files and discovering the differences between them. Two forms of output are generated by SRCCOM: a marked listing and differences. The marked listing is a listing of the first (usually newer) file with 2 asterisks in the right margin after each line which is different from that of the second (usually older) file. An option exists to list only pages which have been changed. Changed pages include those which are renumbered due to inserting or deleting a page. The first page is also always listed. Marked listings with giant pages are not possible.

SRCCOM provides a facility for automatically maintaining a set of
listings. A file named "RECORDOFLISTINGS" contains a record of
the last listed versions of a set of files. SRCCOM uses this
record to determine which files have newer versions and generates
a listing of changed pages and differences for the new version
against the old. When the listing of all such changes is
completed, the record is updated to reflect the newly listed
files. The file "RECORDOFLISTINGS" can be initialized by using
the EXEC'S directory command and putting the output in a file
(the first two lines of heading information must be deleted with
TECO).
#
When SRCCOM starts, if the file "RECORDOFLISTINGS.;1" is present
in the connected directory, SRCCOM will ask the question
"SPECIAL?". A reply of "Y" will cause the file
"RECORDOFLISTINGS" to be ignored. A reply of "N" will cause the
record to be used. A reply of "F" will perform the same
functions as "N" except that any errors in the "RECORDOFLISTINGS"
file will be paused for giving the user a chance to correct the
error. (This will occur when SRCCOM looks for an explicit
version of a file and doesn't find it). The rest of the
initialization dialogue is self explanatory.

TAINT

This writeup describes a very early version of a program called TAINT. TAINT's function is to read files produced by the TENEX MINI-DUMPER program, and to either produce a directory of the tape or to write the files on a DECsystem10 disk system (or both).

The raison d'etre of TAINT is to allow installations running a DECsystem10 monitor to read TENEX dump tapes from co-operating TENEX sites or in preparation for installing a TENEX system of their own. Since the testing of this program has been done under TENEX's 10/50 compatibility system, reports of problems under a real 10/50 system are not impossible and will be appreciated.

TENEX directory names are alphanumeric strings. Therefore, a set of translations to DECsystem10 project - programmer numbers is required, and TAINT sets this up before reading any files (see below). In addition, TENEX file names and extensions are character strings of up to 39 characters, and an 18-bit version number is a part of the file name (not an attribute). When these files are copied to the DECsystem10 disk, the file name must be six or fewer characters, and the extension three or fewer. If this results in truncation of a name, a comment will be typed mentioning the (full) file name. The DECsystem10 version number is not a part of the file name and it is (at present) lost. In particular, if more than one version of the file exists on the TENEX tape, then the one appearing first on the tape will be written and (unless the "SAVE EXISTING FILES" option is used (q.v.)), a version appearing later on the tape will over-write it. Since MINI-DUMPER tapes normally have the highest version first, this would be undesirable, since the lower version number (older) file would remain on the disk.

Program Operation:

When TAINT is started, it carries on the following dialogue with the user.

1. TAINT asks: "LIST TAPE'S DIRECTORY? (Y OR N)" and the user replies with Y or N. If Y, TAINT asks for a file name to write the directory on (defaults being DSK: TENEX.LST). Examples are "LPT:" or "TAPE.DIR".

2. TAINT asks: "READ ANY FILES?"
The user replies Y or N. If Y, TAINT asks for the TENEX DIRECTORY NAME to PROJECT, PROGRAMMER NUMBER correspondence. The simplest response is "carriage return" abbreviated in this memo as "CR". This response implies "Read all TENEX directories on the tape into my directory (the one I am logged in as)". The next simplest response is, for example,
SMITH"CR"
JONES"CR"
"CR"
which means to load all of the files from SMITH and JONES into "my" directory.

Another possibility is:
SMITH,10,7"CR"
"CR"
which loads SMITH's files into [10,7] (assuming, of course, that you have write access to the [10,7] directory). The full case is:
SMITH"CR"
JONES,10,7"CR"
D-DUCK"CR"
M-MOUSE,10,6"CR"
GOOFY"CR"
"CR"
which loads SMITH into "my" directory, JONES and D-DUCK into [10,7] and M-MOUSE and GOOFY into [10,6].

In all of the above, note that the end of the input is determined by typing a blank line.

3. TAINT next asks "SAVE EXISTING DUPLICATE FILES?"
A Y answer implies that no existing files with identical file names (after any truncation) will be overwritten. Therefore, the first (usually newest) file of a given name and extension on the tape will not be written over by a subsequent one with a different version number. Unfortunately, an already existing file on the disk (from an earlier magtape, for example) will not be overwritten either.

An N answer implies that all files will be overwritten, possibly resulting in an older version being the last one written as described above.

To get the latest version from tape, write only into empty (but existing) directories, and answer Y to this question.

Files do not (yet) get their correct creation date from the tape. They just get the date that the tape is written onto the disk, i.e., the current time and date.

4. TAINT asks "TAPE DRIVE NUMBER". The answer to this is a digit from Ø thru 7, for MTAØ: thru MTA7:. (Don't type this digit until the tape is mounted and ready.)

5. TAINT then processes the tape, mentioning on the user's TTY the tape identification, each user name for which files exist (and where they are copied to if they are copied), truncated file names (if any), and any errors encountered. All output files (at present) are a multiple of 1ØØØ words octal in length (a TENEX page).

TAPCNV

TAPCNV is a subsystem which reads a card image file previously processed by MTACPY. It then does the appropriate conversion to ASCII from BCD (Ø26 keypunch code), or eight-bit EBCDIC, or six-bit EBCDIC (Ø29 keypunch code) or from BCL. (Burroughs' external code).

When a file is copied from magtape to disc using MTACPY, an additional auxiliary file is created with extension .RECSIZ which contains the size of each record. TAPCNV uses this auxiliary file to correctly "parse" the data image file into card images for code conversion to ASCII.

On startup, TAPCNV asks:

TAPE IMAGE FILE =

and the correct response is a file previously moved from magtape to disc by MTACPY. Then the program attempts to open the corresponding .RECSIZ file. If it fails to find this file, it asks for it:

RECORD SIZE FILE =

If this file does not exist, the file should be re-acquired from magtape using MTACPY to create this auxiliary file.

TAPCNV then asks,

OUTPUT FILE =

and the correct response is any writeable, ASCII file. If just an altmode is typed for name recognition, the program creates a default file with the same device, directory, and name as the input file, but the extension is .ASCII.

Next, the program asks,

BCL or BCD or EBCDIC?:

conversion begins immediately if the response is BCL or EBCDIC. If the response is BCD, the program asks:

Ø26 or Ø29 CODE?:

and conversion begins once the user responds.

Trailing blanks are suppressed during conversion, a carriage-return, line feed is supplied for each card image, and each new record is assumed to begin a new card image.

```
@TAPCNV
TAPE IMAGE FILE = FILE1
OUTPUT FILE = <altmode>FILE1.ASCII [NEW FILE]_
BCL OR BCD OR EBCDIC?:BCD
Ø26 OR Ø29 CODE?: Ø26
DONE.
@
```

TENEX TECO

This tutorial is designed for a single fast reading. Its purpose is to convey the details of TECO to a reader who is acquainted with the TENEX System and who has some idea of what to expect from a text editor.
#
The first section describes a subset of TECO commands which are sufficient to do any ordinary editing job. The second section describes commands which are essential for editing large files, and includes some of the most useful commands in TECO. The final section gives brief descriptions of the commands of TECO not already described, and introduces the reader to groups of specialized commands for which he may later have a need.
#
For more information see the TENEX TECO Manual.
#
1. BASIC EDITING
#
#
The text which TECO edits is a pure character string, without restrictions or exceptions. A sequence of characters of almost unlimited length can be accommodated, and any one of the 128 ASCII characters can appear at any position in this sequence. Even printer-control operations such as carriage return or tab are represented by ASCII character codes, and they require neither a special form of data to store them nor special commands to manipulate them.
#
The Character Set
#
The complete ASCII Character set is described in Appendix B of the TECO Manual. The characters can be divided into three groups, as follows:
#
-- Each of the graphic characters (95 of them) prints a single letter, digit, or punctuation mark and then advances the printing element one position. The Space character is included in this group.
#
-- Each of the format control characters (7 of them) causes the output device to take a special action, such as starting a new line, spacing to a tab stop, or ringing the bell.
#
-- Each of the command control characters (26 of them) can be used by TENEX software for a one-character, special-purpose command.
#

Names for Control Characters

Since a control character does not type a particular graphic, such as "a" or "+", it must be given an artificial name. In the text of the TECO Manual, a name is used which indicates the mode of production of the character on a terminal. For example, the character produced by holding down the CTRL key and striking the "D" key is called Control-D. In the listings of interactive dialog between the user and TECO, a shorter name is used for a control character, and this name is set off from the rest of the dialog by an enclosing pair of parentheses. For example, "(^D)" is used for Control-D.

The following table gives the short names of the most important control characters and suggests the way in which the characters are used in TECO:

(%)	A Carriage Return was transmitted at this point in the dialog. It was used to end a line.
(HT)	A (Horizontal) Tab was transmitted. It was produced by Control-I and caused the terminal to space to the next tab stop.
(^C)	A Control-C was transmitted. It interrupted TECO.
(^D)	A Control-D was transmitted. It terminated the character string argument of a TECO command.
(ESC)	An Escape was transmitted. It was produced by a key with "ESC" or "ALTMODE" or "PREFIX" written on it and caused TECO to execute a string of previously typed commands.
(DEL)	A Delete was transmitted. It was produced by a key with "DEL" or "RUBOUT" written on it and caused TECO to abort the command it was executing.

The functional descriptions just given are introductory sketches. Subsequent sections will describe the use of these characters in detail.