ROFF4, V1.60

by Ernest E. Bergmann
Physics, Building #16
Lehigh University
Bethlehem, PA 18015

ROFF4, V1.61

by Conrad Kwok
20 3rd St.,Section M,
Fairview Park,
Hong Kong.

ROFF4 is an expanded version of ROFF, based on the formatter in Kernighan and Plauger's book SOFTWARE TOOLS, is written in BDS C. Now it is adapted on IBM PC using Microsoft C. Some of the input files may be used to "set-up" the formatter for a particular style and for particular hardware. It is possible to substitute keyboard input instead of files for educational and test purposes.

95% of the code of ROFF4 V1.61 is the same as V1.60. The changes include:

(1) Modification to run on IBM PC using Microsoft C V1.04 or Lattice C V1.04. Most of the other C compiler should be able to compile the program with minor modification.

(2) Implementation of macro commmand with paramters.

(3) Fixing some minor bugs.

(4) Allow changing of character size on the same line with superscript and subscript.

ROFF was provided by Neal Somos for the public domain via the BDS 'C' Users' Group's volume, CUG -- "Just Like Mom's". Some of this documentation started there as well.

This formatter contains features important for the preparation of technical manuscripts. Special symbols or fonts that can be defined by or for the user can be produced (if the hardware is capable!). Super and subscripts can be handled as well as backspace even for printers without reverse scrolling or backspacing hardware capabilities. However, the output device should recognize separately the <CR> and <LF> functions. The Epson MX-80 with GRAFTRAX 80 was used by the author for most of the development, however he also was able to use the video display of the Exidy Sorcerer, which has user definable graphics. "Preprocessor" directives can be used to merge stock phrases, boiler plate, make macro definitions, automate numbering, and create diversions (for footnotes, table of contents, etc.)

To support the capabilities of WORDSTAR[tm by MicroPro] extended underlining, strikeout, and multiple strike capability are provided as well.

Sample calls:

```
A>roff4 filename1 filename2 filename3 +)
```

this would send the formatted version of these three files to the console and to the printer

```
A>roff4 filename1 +)&filename2
```

this would format filename1 and send it to the printer, console, and to filename2.

```
A>roff4 -s -f filename1 -b filename2 -f -m -r -d -i -g -*
```

The option, -s, causes the formatter to stop (pause) at the start of each page of output; the bell at the console is sounded (if it exists!) and the program waits until any key is pressed at the console. It is essential for printers that are feed single sheets at a time!

I do not use the redirection output feature of DOS 2.0 or higer because this feature is extremely slow (30 times slower) and the output will contain some unwanted characters. Hence the symbol '&' is used for specifying the output file name.

An option that was not shown above, -o[page or range], is used to selectively generate output of ONLY certain pages. It is useful to retype pages that got "eaten" by the printer (Henry Harpending's aptly put language). To retype only page 23, say, make the option: -o23 To retype pages 23 through 29 use: -o23-29 To retype pages 23 to the end use: -23- These options changes the values of the internal variables, FIRSTPAGE and LASTPAGE which originally have the values of 1 and 30000, respectively. Normally this option would be placed early enough in the command line that no pages have been printed yet.

The option, -f, would introduce a formfeed (0CH) into the output stream (useful for placing blank pages, or aligning printer pages) where it appears; in this example, before the first page of output, and, again, at the very end of the output.

The option, -b, turns on the "debug" flag so as to print out lots of diagnostics to the STDERR[console]. Probably only useful for those who are trying to trace the operation of the formatter for elusive bugs". This option is usable only when DEBUGON is defined during compilation. See the file "ROFF4.H"

The option, -m, causes a list of macro definitions to be typed to the console. It is a useful tool for debugging complex macro packages where the preprocessor's expansions are too subtle for humans.

The option, -d, causes a list of diversion files to be typed to the console. Its main virtue is to remind the user what files are being generated and their approximate size.

The option, -i, causes a list of string insertions to be typed to the console. Useful for macro writers, as was the -m, described above. Also, for noting what are the settings of "standard substitutions", such as "today's date".

The option, -r, causes a list of number registers to be typed to the console. Could be useful to find the number of footnotes, etc.

The option, -g, causes a glossary of defined translated characters to be printed on the output device. It is useful to check the appearance of all special definable characters and to produce a "wall chart" of special characters available.

The default option, -*, (the * could be any unassigned option) means keyboard input (buffered line-by-line with a prompt with the character used in the option, here *). Typing a control-Z indicates an end-of-file; the formatter will continue with the next named file. It is intended as a learning aid since one can tryout "tricky" input such as equations. As with standard CP/M, a control-P can be used to toggle the printer to display output that would normally be sent to the console; also, one can edit the keyboard input with the backspace key.

Using ROFF4, you can make nice printouts of a file, with as little or as much help from the program as you want, depending on the commands. There are default values for all parameters, so if you don't put any commands in at all, your file will come out with filled, right-justified lines. The default line-length is 60 characters; the default page-length is 66 lines per page. "Filled lines" means that as many input words as possible are packed onto a line before it is printed; "non-filled" lines go through the formatter w/o rearrangement. "Right-justified" simply means that spaces are added between words to make all the right margins line up nicely. To set a parameter, use the appropriate commands below. All commands have the form of a period followed by two letters. A command line should have nothing on it but the command and its arguments (if any); any text would be lost.

Extra space will separate text sentences. The sentence is recognized by a trailing ':',';','!','?', or a '.'. For the '.' there is the additional requirement that either two or more spaces must spaces must follow it, or that it is at the end of the source line.

A command argument can be either ABSOLUTE or RELATIVE :

```
 .in     5       sets the indent value to 5 spaces
 .in     +5      sets the indent value to the CURRENT value+5
 .ls     -1      sets the line spacing to the current value-1
```

Also, all commands have a minimum and maximum value that will weed out any odd command settings (like setting the line spacing to zero, for example. It won't let you do that, but it could be changed if you REALLY have a burning desire to do

so).

Some commands cause a "break", which is noted in the table below. Before such a command goes into effect, the current line of text is put out, whether it is completely filled or not. (this is what happens at the end of a paragraph, for example.) A line beginning with spaces or a tab will cause a break, and will be indented by that many spaces (or tabs) regardless of the indent value at that time (this is a "temporary indent", which can also be set explicitly). An all blank line also causes a break. If you find that some lines that are indented strangely, and it's not obvious WHY, look at which commands are causing a break, and which aren't. For instance:

```
 .fi
 .ti 0
 this is a line of text
 .in 8
                                         <- blank line
 more text for the machine to play with
```

At first glance it seems obvious that the line "this is a line of text" will be indented zero spaces, but it won't - it will be indented 8. The indent command does NOT cause a break (although the .ti does) so it will not cause the line to be put out before setting the indent value to 8. Then, when the blank line is encountered, it will cause a break - and "this is a line of text" will be indented incorrectly. The above example will give the following lines.

```
        this is a line of text

        more text for the machine to play with
```

It is worthwhile considering placing a ".br", the break command, before each use of ".in"; should future versions of ROFFn have the break already part of the indent command?

Certain system variables are "stacked" to enable reversion to earlier environments instead of "hardcoded" defaults. For example:

```
.ls 1
 .
 .
.ls
```

The first command will produce single line spacing (which is the default, but which may have been set otherwise at the beginning of the manuscript). The second command causes resumption of the original line spacing (either the default or whatever had been chosen previously). Stacked variables include: linespacing, indent column, right margin, translation flag character, page length, top and bottom margin sizes, unexpandable space character, output width, tabsize, and control flag character.

********************** Table of Commands ********************

Command	Break?	Default	stacked	Function
.. string	no			string is "mere"comment
.ab	no			immediate abort back to system
.bj	yes			break with right justification (current line only)
.bp n	yes	n = +1		begin page numbered n
.br	yes			cause a break (this line is not justified)
.cf c	no	c = '^'	Yes	to be used as a prefix to a character that controls print functions such as ^+,^- might be used to bracket superscripts, somewhat like WORDSTAR(TM).
.ce n	yes	n = 1		center next n lines
.db n	no	n = 0(NO)		set debug flag 1 for diagnostics

.di name	no	JUNK.$$$	diversion file (see "PREPROCESSOR")
.dm name	no		define (multiline) macro ("PREPROCESSOR")
.ds /../../../	no	null string	define string replacement ("PREPROCESSOR")
.ed	no		end diversion (see "PREPROCESSOR")
.ef /../../../	no	blanks	even footer titling
.eh /../../../	no	blanks	even header titling
.em			end macro (see "PREPROCESSOR")
.fi	yes		start filling lines
.ff n	no	n = 1(yes)	initially, formfeeds are "off". Can turn them on. Each page then terminated with one formfeed.

.fo /../../../	no	empty	sets both even and odd page footers
.fr # base - ; - - - - .	no	1,no action	defines how to put output device in mode for fractional line spacing(for super,subscripting);see details given below. Complements .WH, below.
.he /../../../	no	empty	sets both even and odd page headers
.ic c	no	c = '\' Yes?	to specify the character used for macro preprocessing to denote the token that follows.
.ig string	no		"ignore" string(see ..)
.in n	no	n = 10 Yes	set indent value to n
.ju	no	initially on	turn on right justification (only applicable if "filling" also)

.ls n	no	n = 1	Yes	set line spacing to n
.m1	no	n = 2	Yes	set topmost margin to n
.m2	no	n = 2	Yes	set 2nd top margin to n lines
.m3	no	n = 2	Yes	1st bottom margin to n lines
.m4	no	n = 2	Yes	bottom-most margin to n lines
.ne n	no/yes	n = 2		"need" n lines; if have them no action; else begins new page
.nf	yes			stop filling lines
.nj	no	initially is justifying		turn off right justification (only relevent if "filling" also)
.of /../../../	no	empty		odd page footer title
.oh /../../../	no	empty		odd page header title
.ou base - - ; - - - .	no	not applicable		direct output of code sequences to output.

.ow	no	n = 80		sets output width for header and footer title
.pc c base - ; - - - .	no	not applicable		used to create definitions for special printer controls, such as for italics.
.pl n	no	n = 66	Yes	sets page length to n
.rg name n	no	n=0	No	create or modify register variable (see "PREPROCESSOR")
.rm n	no	n = 70	Yes	sets right margin to n
.sa string				"say": message to console; like a comment but displayed to operator during run.

.sc c	no	blank	Yes	space character; the visible character that will be trans- literated to unexpanded blank.
.so filename	no	ignored	Yes	reads named file into input stream; cannot be invoked from keyboard input.
.sp n	yes	n = 1		space down n lines
.st n	no	n=1(yes)		stop(pause)at each page start; initially off; may also be enabled by the -s option.
.tc c	no	˜	Yes	set translation flag character, see .tr
.tr c base - ; - - - .	no	not applicable		used to create definitions for special character fonts.
.ts n	no	n = 8	Yes	sets value of tab space
.ti n	yes	n = 0		set temp. indent of n
.wh	no	no action		instructs how to resume whole line spacing; complements .FR, above.

Here's what the page parameters look like:

```
        |<--------output width (.ow)------------------->|
         _______________________________________________
 _      |      top margin(m1) - (includes header)       |
|       |-----------------------------------------------|
|       |              top margin 2                     |
|       |-----------------------------------------------|
P       |      :                               :        |
A       |      :<-indent                       :        |
G       |      :                               :        |
E       |      :lots and lots of silly text and:        |
L       |      :other garbage. Get the picture?:        |
E       |          :This is a temp.  indentation:      |
N       |       :                               :      |
G       |       :               right margin -> :      |
T       |       :                               :      |
H       |       :                               :      |
|       |----------------------------------------------|
|       |              margin 3                        |
|       |----------------------------------------------|
|       |  margin 4 - (includes footer,perhaps ff)     |
-       ------------------------------------------------
```

Minimum acceptable values for M1, M2, M3, and M4; if M1 is set to zero, no header will be shown (even if one was declared). Similarly, if M4 is set to zero, no footer will be displayed.

To change the default for any parameter, simply alter ROFF4.H and reExecute COMPILE.BAT.

```
************************************************************
A Few Extra Comments on Some of the Commands:
************************************************************
```

If you want to center lots of lines, but don't want to count them, do something like this:

```
 .ce     1000
 lots and
 lots of words to
 be centered
 .ce 0
```

will give the lines:

```
                          lots and
                      lots of words to
                         be centered
```

A new paragraph may be caused by using the temporary indent command, like

```
 .ti +5
```

or by simply beginning the paragraph with a tab, as you would if you were just typing.

For special cases, where you wish to place the last "word" at the right-hand margin, such as numbers of equation,

for example:
```
                          x = y+z                        (12)
```
you could input:
```
# x#=#y+z (12)
.bj
```
because we force a break with justification (.bj) of the line with only "three words". (The # is assumed to be the "space character" set up with a .sc command).

Headers and Footers.

A page number can be incorporated into any header or footer by putting a "#" in the title where you want the number to go:
```
 .he     /This is a witty header title for page #/
```
Each time this is printed at the top of a page, the current page number will be substituted for the "#". Each footer and header are ".in 0", even if text is indented.

Headers and footers are in three parts. These parts are left justified, centered, and right justifed. Any of these three parts may be left out. The right justification is fixed to the margin that is set by the .OW command. One may pick the headers and footers separately for even and odd pages. For example, one could place even and odd page numbers at the bottom outside of each page by:
```
 .ef /Page #///
 .of ///Page #/
```

Any printable character, here the '/', can be used to delimit the three strings that make up the titles, so long as it is not the "insert character" (usually, '\') and is not present in any of the three strings.

The program can be made to wait for the operator to load single sheets of paper by the -s option and/or by the command: .st

----

If you want to send the output to a file, and don't want the page breaks in there set margins 1-4 to zero.

Where you need to supply code sequences for immediate or for subsequent output (for .ou or for .tr) one needs to supply the number base (binary, octal, decimal, or hexadecimal) by supplying a token that begins with (upper and lowercase are both o.k.): b, o(or q), d, or h. Following the base on the same and/or subsequent lines one supplies the codes that will form the "code string". These codes are delimited by white space (not commas!) and the sequence is eventually terminated by a token beginning with a period. The ends of any of these lines may contain comments if they are set off by white space and a semicolon. For examples:

```
 .ou hex 11 1C 8C 0
 99 6C 55 ;get ready!
 .end

 .TR = binary ;"identity"operator (triple equal sign) on MX-80
 00011011 ;ESC
 01001011 ;4B
 00000110 ;6 bit patterns
 0        ;follow:
 00101010 ;left top,bot
 00101010
 00101010
 00101010
 00101010
 00000000 ;right top,bot
 .en
```

The first of these examples might be used to get some strange printer to cooperate in standing on its head or something.

The second, lengthier example is taken from a file, MX, that defined quite a number of special characters for the MX-80 with GRAFTRAX 80. Because that printer uses dot graphics, I chose to make the definition in binary so that the placement of the individual dots is easier to visualize. Later, in the text the combination: ˜= will cause the printer to be sent this code so that it will print the specialized symbol.

SPECIAL SYMBOLS

If one includes a set of definitions such as in the MX file, one can specify the use of special symbols, which can be chosen to fit the application (and the hardware!) For example, the MX-80 printer equipped with GRAFTRAX 80 can accept dot addressed graphics. [See the MX file for examples with this hardware]. Other printers may be able to simulate symbols by a combination of overstruck characters. Still other output devices may be capable of displaying desired special symbols by use of the "parity bit".

The .tr and .tc commands define the codes for the special symbols and the translation character. To create a "wall chart" that lists the special characters on the output device you could use the -g option on the command line, for example:

```
A>roff4 mx -g )
```

====================================

PRINTER CONTROL

It is assumed that the output device can accept carriage returns and will not linefeed in the absence of the linefeed character. Using this assumption, super- and subscripting, backspacing, underscoring, strickout, and multiple impressions are supported in a manner somewhat analogous to WORDSTAR [tm by MicroPro], but more generally.

If the printer can be placed in fractional linespacing mode, so much the better, as full line spacing for super- and subscripts does not look as "natural". Also, with the half line spacing, one can build up larger characters (such as summation and integration symbols) since some overlap does occur. To implement such fractional spacing one uses the commands, .fr and .wh (probably at the beginning of the input file, along with other information relevent to the output device and style). These commands describe the operational codes sent to the output to switch the printer to FRactional spacing and back to WHole line spacing.

For example, I use for the MX-80 printer equipped with GRAFTRAX 80 the following:

```
 .WH HEX 1B 32 . ; 6 lines/inch is standard
 .fr 2 hex 1b 33 12 . ; 18/216" = halfline spacing
```

The initial 2 in .fr tells the formattter that 2 fractional [half] lines are equivalent to a conventional whole line. The original description of the required codes were in hexadecimal, so I kept matters as simple as possible by using the same number base so that I would not make any conversion mistakes!

The printer control requests are embedded in the text; they are NOT set off in separate lines as the "dot" commands are set apart. Each request is made up of two printable characters, the first of which is the "control flag character" (the default is '^'). Here is a table of control functions presently supported by ROFF4, version 1.30: (additional codes can be created with the "printer control" command, .pc)

^+	up a fractional line; may be used several times to increase vertical rise. [used at start of a superscript and at the end of a subscript]
^-	down a fractional line; may be used several times to

increase vertical drop. [used at the start of a subscript and at the end of a superscript]

^h,^H backspace one character column. Do NOT backspace over ordinary blanks ("unexpandable" space is o.k.) if you are in "fill" mode.

```
^(,^)    Note current column position; return to noted position.
^[,^]    "      "      "      "      "      "      "
^{,^}    "      "      "      "      "      "      "
```

The above three pairs of controls are often more convenient then multiple, explicit backspaces, ^H, especially for "built-up" fractions and matrices.

^B,^b Start, end boldface (increase, decrease the number of impressions by a factor of 3).

^D,^d Start, end doublestrike (increase, decrease the number of impressions by a factor of 2).

^U,^u Start, end underscore (will not underscore expandable white space; will ride up and down with super and subscripts.)

^X,^x Start, end strikeout (similar to underscore, above, but overprints with '-' instead of underlines).

Note that the last 4 pairs are "case sensitive"; namely, the uppercase starts some activity, whereas the lowercase equivalent sqelches it; these controls are NOT "toggles".

An involved example of the use of printer controls would be to create a 3 by 3 matrix:

```
MATRIX =#^+^+^(|1#2#3|^)^-^-|4#5#6|^)^-^-|7#8#9|^+^+
```

which should produce (with a half-spacing) printer:

```
         |1 2 3|
MATRIX = |4 5 6|
         |7 8 9|
```

[the demonstration file, MATRIX, has been provided as a demonstration of the above].

Several points should be observed. There should be no expandable blank spaces if you are in fill mode, otherwise, the result might be very strange! (ROFF4 does some checks to flag such attempts). We are assuming here that the '#' are unexpandable spaces (chosen with the .sc command). The first printable character in the complex, the '=', is at the leftmost edge; the last printable character, the '|' following the '9', is at the rightmost edge of this assemblage. The final height is adjusted (by the trailing ^+^+) to match the initial height. The present limit of the line buffering is 255 characters; I assume that is not too chancy.

One can define additional printer control codes using the .pc command. For example, the MX-80 printer with Graftrax is switched to italics with the sequence <ESC> '4'; and italics are turned off with <ESC> '5'. We could define ^I to start italics and ^i to end them:

```
 .pc I hex ;italics on (MX-80   Graftrax)
 1B 34
 .en
 .pc i hex ; italics off (MX-80   Graftrax)
 1B 35
 .en
```

================================

THE PREPROCESSOR

In the following we describe the advanced macro preprocessing features of this formatter which provide users with labor saving tools but which are probably not necessary at first. The beginning user may be able to achieve most goals without the "preprocessing", but by using an editor more then otherwise. The more advanced user will begin to appreciate these features more.

In the following discussion we will assume the default insert character, '\', and the default command character, '.', will be used. (It is rare that you should change these anyway!)

The insert character is used to denote where a replacement should be used. For example, in:

```
Today, \date\, is special.
```

the block, "\date\", would be replaced as this sentence is being input. If a prior string definition of the form:

```
 .ds *date*January 1, 1983*
```

had been processed previously then the example, after text substitution, would become:

Today, January 1, 1983, is special.

The string definition command, .ds, expects that the first visible character, here a '*', is the delimiter of the start and end ot the two parts in the definition; any printable character (that is not present in either string!) may be used.

If no string definition had been provided for "date", the user will be prompted while the formatter is trying to input this sample line. The console will get some message like:

```
[Bell]Please define <date>:
```

Whatever you type in will be used to form an "effective" .ds definition. This feature should be useful in applications where information should be changed or updated each time the formatter is run, such as today's date, the addressee's name and address in a form letter, etc. A sample file, FORM is included to demonstrate both of the above means to define string substitutions.

An important restriction must be observed when using "definitions on the run". They must not be first used inside of multiline definitions (namely inside of .ou, .tr, .dm, .pc, .wh, and .fr) because the building of both definitions will cause them to interfere with each other. ROFF4, v1.6 will test for such contention and abort operation if one is found. An example of such a situation and its remedy is shown below:

```
 .sa chose 0 for DRAFT and 1 for CORRESPONDENCE
 .ou hex
 1B
 3\font\
 .end .ou
```

This example, which might be used to initialize the Okidata Microline 92 printer to go into correspondence quality or into draft quality printing would cause problems if "font" is supposed to be defined here during execution. We are in the midst of defining an output string for the printer (ESC "0" or ESC "1") when we are asking ROFF4 to create (simultaneously) a definition for "font"; the program will abort rather than continue with the two definitions mangling each other. Here is a modified version of above without the problem:

```
 .sa chose 0 for DRAFT and 1 for CORRESPONDENCE
 .. this comment containing \font\ is "ignored"
 .ou hex
 1B
 3\font\
 .end .ou
```

The fix here is that the formatter will encounter "\font\" in the comment and complete a definition for "font" before takling the .ou command; no simultaneous definitions, no problems!

Similar to string definitions are register variables, which are created and modified with the .rg command. Variables are useful for enumeration such as equation numbering:

```
 .rg eqnum 1
```

would create a register named "eqnum" with the current value of 1. The the text might use it with, say:

```
         x = y+1 (\eqnum\)
```

which would be converted on input into:

```
         x = y+1 (1)
```

A subsequent instruction:

```
 .rg eqnum +1
```

would take the current value of "eqnum" and increase it by 1 (so that it would now be 2 in our example:

```
         a = b+c (\eqnum\)
```

would become:

```
         a = b+c (2)
```

There is a special, reserved insertion, \#\, which will provide the current page number. It should prove useful in setting up tables of contents (see "diversions", below). Trivial examples of its use are to be found in the files, BPTEST and MARGINS. In rare cases it may be off one page because it may be read while between pages; how can one handle the sentence, "This sentence is on page XXX," when the sentence straddles two pages?

Since we have defined a special register name, '#', we should comment on what happens if you create a register instruction with that name, such as:

```
 .rg # +1
```

You will be changing the value of the page number of the FOLLOWING pages. This is useful for leaving gaps in the pagination for later inclusion of full page illustrations. This feature is demonstrated (tested) in the file, MARGINS. I wish to thank Henry Harpending for suggesting this.

The insert character has other properties. The insert character can be placed into the input by repeating it, namely, "\\" becomes "\". (useful for delaying substitutions). For example, defining:

```
 .ds 'EN'(\\eqnum\\)'
```

will identify "EN" with "(\eqnum\)" and so our equation example above could have been:

```
         a = b+c \EN\
```

Delaying the evaluation of "eqnum" until EN is invoked (instead of when it was defined) means that the proper numbering of equations will occur instead of wrongly supplying the value of "eqnum" from the time that EN was first created.

If the insert character is at the end of a line, it negates the following newline sequence; thus the next line is merged with the current line. For example:

```
antidisestab\
lishmentarianism
```

is equivalent to:

```
antidisestablishmentarianism
```

"Macro" definitions are used when we wish to identify several lines with an insertion. Such definitions are created with the .dm ["define macro"] and completed with the .em ["end macro"] commands. For example, we might wish to use the following sequence over and over again at the start of paragraphs:

```
 .sp 1
 .ne 2
 .ti +5
```

to separate the paragraphs by blank lines, keep them from starting excessively close to the bottom of the page, and indenting them 5 spaces to the right of the current left margin. We might want to define the "command" as "paragraph" [personally, I might call it "P", because it would be used a lot and my typing ...]:

```
 .dm paragraph
 .sp 1
 .ne 2
 .ti +5
 .em
```

Subsequently, whenever we wished to start a paragraph we would creat a command line:

```
 .paragraph
```

instead of more tediously creating every time the three commands we mentioned above.

The names of all macros, strings, and number registers are "case sensitive". That is to say that capitalization and/or lower case are distinguished and, say,

```
 .Paragraph
```

Would not be recognized as the same as the sample macro we just defined. However, all the "built-in" commands, those which were listed in the command table, are not case sensitive and are recognized on the first two letters alone, even if arbitrary letters or numbers follow immediately. If we had a line:

```
 .time
```

it would be identified with a "time" macro definition, if one had been created; It would not be confused with a "Time" macro definition. If there is no "time" macro, then it would be matched with the "built-in", .TI ["temporary indent"].

A macro command may contain parameters(s). For example

```
 .dm HEADING
 .sp 2
 ^B$0^b
 .sp 2
 .em
```

defines a macro for printing heading. When the line ".HEADING Heading" appear on the beginning of a line, the word "Heading" will be printed in boldface with two blank lines above and below. i.e.

Heading

The maximum number of parameters for a macro is 10 ($0-$9). The parameter(s) must be on the same line of the macro call. The parameters are separate by a non-alphanumeric character except '+' and '-' which appear as the first character of the parameter line. If the first character is a alphanumeric character, then white space will be assumed as the delimiter. An example can be found above. If the require parameter(s) is/are not defined by the macro call, then the parameter(s) will be treated as null string.

There is another object formed and used somewhat like a macro; it is called a "diversion (file)" and is useful for making lists such as references [footnotes] and tables of contents. A diversion is created or continued with the commands: .DI [diversion] and .ED [end diversion]. A diversion can grow to be quite large and is, in fact, a disk file. To "regurgitate" the diversion file, its name can be placed in the original command line, along with the other input file names; alternatively, files can be retrieved with the .SO ["source"] command. The advantage of using .SO is that inclusion can be accomplished without a page break, nor even a line break between input files. The
.SO command is like a "CALL" or "GOSUB" in that there can be nested .SO invokations; one can access a file with .SO that contains in turn a .SO command, etc. It is a limitation of ROFF4 at present to not be able to handle the .so command from keyboard input (it could be useful). The files, SOTEST, ONE, TWO, and THREE are provided to test and demonstrate the .SO command.

All file names referenced by .di and .so are automatically treated as uppercase. The naming conventions should conform to the operating system (CP/M). It is a limitation of the formatter at present to not realize that "A:ZZ" would be the same as "ZZ"; be sure to use the same form throughout!

We shall present a detailed example of the use of the above preprocessing commands to automate footnote and reference numbering and collection.

We start by creating a register variable, "f#", to keep track of the current footnote number:

```
 .rg f# 1
```

We shall use, say, "[15]" as our means to display reference numbering. (We could have used superscripts instead with "^+15^-"):

```
 .ds "fn"[\\f#]"
```

We have used "\\" so that "fn" is defined as "[\f#\] and will be evaluated with the current footnote number at the time of use (not of the time we nade this .ds definition). By typing

\fn\ we will get the reference in the form, "[number]", that we wanted.

We want to create a diversion, "FNOTES", into which we will place all our references. The head of this file will be titled with "REFERENCES":

```
 .di fnotes
 .ls 1
 .sp 1
 .ce 1
 REFERENCES
 .sp 2
 .ed
```

The diversion will contain (hopefully) a list of numbered footnotes. To make the addition of these footnotes as painless as possible, we define two macros, "FS" [footnote start] and "FE" [footnote end]:

```
 .dm FS
 .di fnotes
 .sp 1
 \\fn\\\\
 .em
         and:
 .dm FE
 .ed
 .rg f# +1
 .em
```

The FS macro skips a line and attaches the evaluation of \fn\ to the start of the line that follows the macro during execution. The lines that follow the FS macro will be diverted to FNOTES. The FE macro terminates the diversion and, also, increments the footnote number, f#.

We could try a very small piece of text now:

```
 .nf
 It is a nice day.\fn\
 .FS
 conventional expression.
```

```
 .FE
 It's a crummy day.\fn\
 .FS
 unconventional!
 .FE
```

The formatter will generate:

```
 It is a nice day.[1]
 It is a crummy day.[2]
```

and the diversion file, FNOTES, will contain:

```
 .ls 1
 .sp 1
 .ce 1
 REFERENCES
 .sp 2
 .sp 1
 [1]conventional expression.
 .sp 1
 [2]unconventional!
```

which, after formatting, will be:

```
                         REFERENCES



[1]conventional expression.

[2]unconventional!
```

→