

PRO/VENIX™

UNIX™

for

Professional™

Document Processing Guide

digital
software

PRO/VENIX™
for the Professional

Document Processing Guide

Developed by:

VenturCom, Inc.
215 First Street
Cambridge, MA 02142

Digital Equipment Corporation
Maynard, MA 01754

First Printing

The software described in this manual is distributed as part of Digital Equipment Corporation's Digital Classified Software (DCS) Program. This program enables software developers to submit their software products to Digital for testing according to Digital quality standards for third party software. This software product has met the DCS standard specified in the software product description (SPD) for this product. You should refer to the SPD for information about these standards, the hardware and software required to run this product, and warranties (if any warranty is available).

The software described in this manual is furnished under a license and may only be used or copied in accordance with the terms of that license. This manual is reproduced with the permission of VenturCom, Inc.

Copyright © 1983, by Western Electric. All Rights Reserved.

Portions Copyright © 1984 VenturCom, Inc. All Rights Reserved.

Except as may be stated in the SPD for this product, no responsibility is assumed by Digital or its affiliated companies for use or reliability of this software, or for errors in this manual or in the software. Additional support and/or warranty services may be available from the developer of this software product. Digital has no connection with, and assumes no responsibility or liabilities in connection with these services.

This manual is subject to change without notice and does not constitute a commitment by Digital.

VENIX is a trademark of VenturCom, Inc.

UNIX is a trademark of AT&T Technology, Inc.

The following are trademarks of Digital Equipment Corporation:

DEC	DECwriter	Professional	VAX
DECmate	DIBOL	Rainbow	VMS
DECnet	MASSBUS	RSTS	VT
DECsystem-10	PDP	RSX	Work Processor
DECSYSTEM-20	P/OS	UNIBUS	digital
DECUS			

The PRO/VENIX† Documentation Set

The PRO/VENIX documentation set consists of the following manuals:

PRO/VENIX Installation and System Manager's Guide

The set up and maintenance of PRO/VENIX are described in the installation sections. Other articles explain the UNIX-to-UNIX‡ communications systems. The “System Maintenance Reference Manual” contains reference pages for devices and system maintenance procedures (sections (7) and (8)).

PRO/VENIX User Guide

The *User Guide* contains tutorials for newcomers to PRO/VENIX, covering basic use of the system, the editor **vi** and use of the command language interpreters.

PRO/VENIX Document Processing Guide

The line and screen editors and **nroff**-related text formatting utilities are described in the Document Processing Guide. Topics include: line editor **ed**, and stream editor **sed**; the text formatter **nroff**; the **nroff**- preprocessors **tbl** and **neqn**.

PRO/VENIX Programming Guide

The chapters in the *Programming Guide* explicate the different programming languages for VENIX.

† VENIX is a trademark of VenturCom, Inc.

‡ UNIX is a trademark of Bell Laboratories.

PRO/VENIX Support Tools Guide

This guide includes tools for programming, such as the compiler-writing languages Yacc and Lex, the M4 Macro processor, the program development utility Make, and the desk calculator programs DC and BC.

PRO/VENIX User Reference Manual

This is a complete and concise reference for the PRO/VENIX system. This volume contains write-ups on all PRO/VENIX commands.

PRO/VENIX Programmer Reference Manual

The reference pages in this volume include system calls, library functions, file formats, miscellaneous functions and games.

CONTENTS

INTRODUCTION

- Chapter 1. ADVANCED EDITING ON VENIX†**
- Chapter 2. SED — A NON-INTERACTIVE TEXT EDITOR**
- Chapter 3. USING THE -MS MACROS**
- Chapter 4. NROFF TUTORIAL**
- Chapter 5. NROFF USER'S MANUAL**
- Chapter 6. NROFF TERMINAL DESCRIPTOR TABLE**
- Chapter 7. TABLE FORMATTING PROGRAM**
- Chapter 8. MATHEMATICS TYPESETTING PROGRAM**

CONTENTS

1

1.1 INTRODUCTION	1-1
1.2 SPECIAL CHARACTERS	1-1
1.3 LINE ADDRESSING IN THE EDITOR	1-19
1.4 GLOBAL COMMANDS	1-27
1.5 CUT AND PASTE WITH VENIX COMMANDS	1-30
1.6 CUT AND PASTE WITH THE EDITOR	1-34
1.7 SUPPORTING TOOLS	1-40

Chapter 1

ADVANCED EDITING ON VENIX†

1.1 INTRODUCTION

Although VENIX provides remarkably effective tools for text editing, some general knowledge of the VENIX system will assist you to use the editors most effectively. This chapter assumes that you have some familiarity with the **ed** editor. If you are a novice user, consult VENIX FOR BEGINNERS in the *User Guide* for some background information. Further information on all commands discussed here can be found in the *User Reference Manual*.

Topics covered include special characters in searches and substitute commands, line addressing, the global commands, and line moving and copying. There are also brief discussions of effective use of related tools, like those for file manipulation, and those based on **ed**, like **grep** and **sed**.

A word of caution. There is only one way to learn to use something, and that is to use it. Reading a description is no substitute for trying something. This chapter should give you ideas about what to try, and then you can experiment on your own.

1.2 SPECIAL CHARACTERS

The editor **ed** is the primary interface to the system for many people, so it is worthwhile to know how to get the most out of **ed** for the least effort.

The next few sections will discuss shortcuts and labor-saving devices. Not all of these will be instantly useful to any one person, of course, but a few will be, and the others should give you ideas to store away for future use.

ADVANCED EDITING

1.2.1 The List command ‘l’

ed provides two commands for printing the contents of the lines you’re editing. For example, **p**, in combinations like

1,\$p

will print all the lines you’re editing, or

s/abc/def/p

will change ‘abc’ to ‘def’ on the current line and print it. The list command, **l** (the letter ‘l’), gives slightly more information than **p**. In particular, **l** makes visible characters that are normally invisible, such as tabs and backspaces. If you list a line that contains some of these, **l** will print each tab as > and each backspace as <. This makes it much easier to correct the sort of typing mistake that inserts extra spaces adjacent to tabs, or inserts a backspace followed by a space.

The **l** command also ‘folds’ long lines for printing — any line that exceeds 72 characters is printed on multiple lines; each printed line except the last is terminated by a backslash \, so you can tell it was folded. This is useful for printing long lines on short terminals.

Occasionally the **l** command will print in a line a string of numbers preceded by a backslash, such as \07 or \16. These combinations are used to make visible characters that normally don’t print, like form feed or vertical tab or bell. Each such combination is a single character. When you see such characters, be wary — they may have surprising meanings when printed on some terminals. Often their presence means that your finger slipped while you were typing; you almost never want them.

1.2.2 The Substitute Command ‘s’

Most of the next few sections will be taken up with a discussion of the substitute command **s**. Since this is the command for changing the contents of individual lines, it probably has the most complexity of any **ed** command, and the most potential for effective use.

As the simplest place to begin, recall the meaning of a trailing **g** after a substitute command. With

s>this/that/

and

s>this/that/g

the first one replaces the first ‘this’ on the line with ‘that’. If there is more than one ‘this’ on the line, the second form with the trailing **g** changes all of them.

Either form of the **s** command can be followed by **p** or **l** to ‘print’ or ‘list’ (as described in the previous section) the contents of the line:

s>this/that/p
s>this/that/l
s>this/that/gp
s>this/that/gl

are all legal, and mean slightly different things. Make sure you know what the differences are.

Of course, any **s** command can be preceded by one or two ‘line numbers’ to specify that the substitution is to take place on a group of lines. Thus

1,\$s/mispell/misspell/

changes the first occurrence of ‘mispell’ to ‘misspell’ on every line of the file. But

1,\$s/mispell/misspell/g

changes every occurrence in every line (and this is more likely to be what you wanted in this particular case).

You should also notice that if you add a **p** or **l** to the end of any of these substitute commands, only the last line that got changed will be printed, not all the lines. We will talk later about how to print all the lines that were modified.

ADVANCED EDITING

1.2.3 The Undo Command ‘u’

Occasionally you will make a substitution in a line, only to realize too late that it was a ghastly mistake. The ‘undo’ command **u** lets you ‘undo’ the last substitution: the last line that was substituted can be restored to its previous state by typing the command

u

1.2.4 The Metacharacter ‘.’

As you have undoubtedly noticed when you use **ed**, certain characters have unexpected meanings when they occur in the left side of a substitute command, or in a search for a particular line. In the next several sections, we will talk about these special characters, which are often called ‘metacharacters’.

The first one is the period ‘.’. On the left side of a substitute command, or in a search with ‘/.../’, ‘.’ stands for any single character. Thus the search

/x.y/

finds any line where ‘x’ and ‘y’ occur separated by a single character, as in

**x + ^y
x - ^y
x□^y
x.y**

and so on. (We will use □ to stand for a space whenever we need to make it visible.)

Since ‘.’ matches a single character, that gives you a way to deal with funny characters printed by **I**. Suppose you have a line that, when printed with the **I** command, appears as

.... th\07is

and you want to get rid of the \07 (which represents the bell character, by the way).

The most obvious solution is to try

s/\07//

but this will fail. (Try it.) The brute force solution, which most people would now take, is to re-type the entire line. This is guaranteed, and is actually quite a reasonable tactic if the line in question isn't too big, but for a very long line, re-typing is a bore. This is where the metacharacter '.' comes in handy. Since '\07' really represents a single character, if we say

s/th.is/this/

the job is done. The '.' matches the mysterious character between the 'h' and the 'i', whatever it is.

Bear in mind that since '.' matches any single character, the command

s./,/

converts the first character on a line into a ',', which very often is not what you intended.

As is true of many characters in **ed**, the '.' has several meanings, depending on its context. This line shows all three:

.s/. /.

The first '.' is a line number, the number of the line we are editing, which is called 'line dot'. The second '.' is a metacharacter that matches any single character on that line. The third '.' is the only one that really is an honest literal period. On the right side of a substitution, '.' is not special. If you apply this command to the line

Now is the time.

the result will be

.ow is the time.

ADVANCED EDITING

which is probably not what you intended.

1.2.5 The Backslash '\'

Since a period means ‘any character’, the question naturally arises of what to do when you really want a period. For example, how do you convert the line

Now is the time.

into

Now is the time?

The backslash ‘\’ does the job. A backslash turns off any special meaning that the next character might have; in particular, ‘\.’ converts the ‘.’ from a ‘match anything’ into a period, so you can use it to replace the period in

Now is the time.

like this:

s/>\./?/

The pair of characters ‘\.’ is considered by ed to be a single real period.

The backslash can also be used when searching for lines that contain a special character. Suppose you are looking for a line that contains

.PP

The search

/.\PP/

isn’t adequate, for it will find a line like

THE APPLICATION OF ...

because the ‘.’ matches the letter ‘A’. But if you say

/^\.\PP/

you will find only lines that contain ‘.PP’.

The backslash can also be used to turn off special meanings for characters other than ‘.’. For example, consider finding a line that contains a backslash. The search

/\//

won’t work, because the ‘\’ isn’t a literal ‘\’, but instead means that the second ‘/’ no longer delimits the search. But by preceding a backslash with another one, you can search for a literal backslash. Thus

/\\//

does work. Similarly, you can search for a forward slash ‘/’ with

/\\//

The backslash turns off the meaning of the immediately following ‘/’ so that it doesn’t terminate the /.../ construction prematurely.

As an exercise, before reading further, find two substitute commands each of which will convert the line

\x\.\.y

into the line

\x\y

Here are several solutions; verify that each works as advertised.

s/\\\\.//
s/x../x/
s/..y/y/

A couple of miscellaneous notes about backslashes and special characters. First, you can use any character to delimit the pieces of an s command: there is nothing sacred about slashes. (But you must use slashes for context searching.) For

ADVANCED EDITING

instance, in a line that contains a lot of slashes already, like

```
//exec //sys.fort.go // etc...
```

you could use a colon as the delimiter. To delete all the slashes, type

```
s:/::g
```

Second, if # and @ are your character erase and line kill characters, you have to type \# and \@; this is true whether you're talking to **ed** or any other program.

When you are adding text with a or i or c, backslash is not special, and you should only put in one backslash for each one you really want.

1.2.6 The Dollar Sign '\$'

The next metacharacter, the '\$', stands for 'the end of the line'. As its most obvious use, suppose you have the line

Now is the

and you wish to add the word 'time' to the end. Use the \$ like this:

```
s/$/□time/
```

to get

Now is the time

Notice that a space is needed before 'time' in the substitute command, or you will get

Now is thetime

As another example, replace the second comma in the following line with a period without altering the first:

Now is the time, for all good men,

The command needed is

s/\$./

The \$ sign here provides context to make specific which comma we mean. Without it, of course, the s command would operate on the first comma to produce

Now is the time. for all good men,

As another example, to convert

Now is the time.

into

Now is the time?

as we did earlier, we can use

s/>.\$/?/

Like '.', the '\$' has multiple meanings depending on context. In the line

\$s/\$\$/

the first '\$' refers to the last line of the file, the second refers to the end of that line, and the third is a literal dollar sign, to be added to that line.

1.2.7 The Circumflex '^'

The circumflex (or hat or caret) '^' stands for the beginning of the line. For example, suppose you are looking for a line that begins with 'the'. If you simply say

/the/

you will in all likelihood find several lines that contain 'the' in the middle before arriving at the one you want. But with

ADVANCED EDITING

/^{the}/

you narrow the context, and thus arrive at the desired one more easily.

The other use of '^' is of course to enable you to insert something at the beginning of a line:

s/^/□/

places a space at the beginning of the current line.

Metacharacters can be combined. To search for a line that contains only the characters

.PP

you can use the command

/^{^\>.PP\$}/

1.2.8 The Star '*'

Suppose you have a line that looks like this:

text x y text

where text stands for lots of text, and there are some indeterminate number of spaces between the x and the y. Suppose the job is to replace all the spaces between x and y by a single space. The line is too long to retype, and there are too many spaces to count. What now?

This is where the metacharacter '*' comes in handy. A character followed by a star stands for as many consecutive occurrences of that character as possible. To refer to all the spaces at once, say

s/x□*y/x□y/

The construction '□*' means 'as many spaces as possible'. Thus 'x□*y' means 'an x, as many spaces as possible, then a y'.

The star can be used with any character, not just space. If the original example was instead

text x-----y *text*

then all ‘-’ signs can be replaced by a single space with the command

s/x-*y/x□y/

Finally, suppose that the line was

text x.....y *text*

Can you see what trap lies in wait for the unwary? If you blindly type

s/x.*y/x□y/

what will happen? The answer, naturally, is that it depends. If there are no other x's or y's on the line, then everything works, but it's blind luck, not good management. Remember that ‘.’ matches any single character? Then ‘.*’ matches as many single characters as possible, and unless you're careful, it can eat up a lot more of the line than you expected. If the line was, for example, like this:

text x *text* x.....y *text* y *text*

then saying

s/x.*y/x□y/

will take everything from the first ‘x’ to the last ‘y’, which, in this example, is undoubtedly more than you wanted.

The solution, of course, is to turn off the special meaning of ‘.’ with ‘\.’:

s/x\.*y/x□y/

Now everything works, for ‘\.*’ means ‘as many periods as possible’.

ADVANCED EDITING

There are times when the pattern ‘.*’ is exactly what you want. For example, to change

Now is the time for all good men

into

Now is the time.

use ‘.*’ to eat up everything after the ‘for’:

s/□for.*//

There are a couple of additional pitfalls associated with '*' that you should be aware of. Most notable is the fact that 'as many as possible' means zero or more. The fact that zero is a legitimate possibility is sometimes rather surprising. For example, if our line contained

and we said

s/x□*y/x□y/

the first 'xy' matches this pattern, for it consists of an 'x', zero spaces, and a 'y'. The result is that the substitute acts on the first 'xy', and does not touch the later one that actually contains some intervening spaces.

The way around this, if it matters, is to specify a pattern like

/x□□*y/

which says ‘an x, a space, then as many more spaces as possible, then a y’, in other words, one or more spaces.

The other startling behavior of '*' is again related to the fact that zero is a legitimate number of occurrences of something followed by a star. The command

s/x*/y/g

when applied to the line

abcdef

produces

yaybycodyeyfy

which is almost certainly not what was intended. The reason for this behavior is that zero is a legal number of matches, and there are no x's at the beginning of the line (so that gets converted into a 'y'), nor between the 'a' and the 'b' (so that gets converted into a 'y'), nor ... and so on. Make sure you really want zero matches; if not, in this case write

s/xx*/y/g

'xx*' is one or more x's.

1.2.9 The Brackets '[]'

Suppose that you want to delete any numbers that appear at the beginning of all lines of a file. You might first think of trying a series of commands like

```
1,$s/^1*//  
1,$s/^2*//  
1,$s/^3*//
```

and so on, but this is clearly going to take forever if the numbers are at all long. Unless you want to repeat the commands over and over until finally all numbers are gone, you must get all the digits on one pass. This is the purpose of the brackets [and].

The construction

[0123456789]

matches any single digit — the whole thing is called a 'character class'. With a character class, the job is easy. The pattern '[0123456789]*' matches zero or more digits (an entire number), so

ADVANCED EDITING

1,\$s/^[0123456789]*//

deletes all digits from the beginning of all lines.

Any characters can appear within a character class, and just to confuse the issue there are essentially no special characters inside the brackets; even the backslash doesn't have a special meaning. To search for special characters, for example, you can say

/[\.\\$^[]/

Within [...], the '[' is not special. To get a ']' into a character class, make it the first character.

It's a nuisance to have to spell out the digits, so you can abbreviate them as [0–9]; similarly, [a–z] stands for the lower case letters, and [A–Z] for upper case.

As a final frill on character classes, you can specify a class that means 'none of the following characters'. This is done by beginning the class with a '^':

[^0–9]

stands for 'any character except a digit'. Thus you might find the first line that doesn't begin with a tab or space by a search like

/[^^(space)(tab)]/

Within a character class, the circumflex has a special meaning only if it occurs at the beginning. Just to convince yourself, verify that

/[^~]/

finds a line that doesn't begin with a circumflex.

1.2.10 The Ampersand ‘&’

The ampersand ‘&’ is used primarily to save typing. Suppose you have the line

Now is the time

and you want to make it

Now is the best time

Of course you can always say

s/the/the best/

but it seems silly to have to repeat the ‘the’. The ‘&’ is used to eliminate the repetition. On the right side of a substitute, the ampersand means ‘whatever was just matched’, so you can say

s/the/& best/

and the ‘&’ will stand for ‘the’. Of course this isn’t much of a saving if the thing matched is just ‘the’, but if it is something long, or if it is something like ‘.*’ which matches a lot of text, you can save some tedious typing. There is also much less chance of making a typing error in the replacement text. For example, to parenthesize a line, regardless of its length,

s/.*/(&)/

The ampersand can occur more than once on the right side:

s/the/& best and & worst/

makes

Now is the best and the worst time

and

s/.*/&? &! !/

converts the original line into

ADVANCED EDITING

Now is the time? Now is the time!!

To get a literal ampersand, naturally the backslash is used to turn off the special meaning.

s/ampersand/\&/

converts the word into the symbol. Notice that ‘&’ is not special on the left side of a substitute, only on the right side.

1.2.11 Substituting Newlines

ed provides a facility for splitting a single line into two or more shorter lines by ‘substituting in a newline’. As the simplest example, suppose a line has gotten unmanageably long because of editing (or merely because it was unwisely typed). If it looks like

text **xy** *text*

you can break it between the ‘x’ and the ‘y’ like this:

**s/xy/x\
y/**

This is actually a single command, although it is typed on two lines. Bearing in mind that ‘\’ turns off special meanings, it seems relatively intuitive that a ‘\’ at the end of a line would make the newline there no longer special.

You can in fact make a single line into several lines with this same mechanism. As a large example, consider underlining the word ‘very’ in a long line by splitting ‘very’ onto a separate line, and preceding it by the **roff** or **nroff** formatting command ‘.ul’.

text **a very big** *text*

The command

```
s/□very□/＼  
.ul＼  
very＼  
/
```

converts the line into four shorter lines, preceding the word ‘very’ by the line ‘.ul’, and eliminating the spaces around the ‘very’, all at the same time.

When a newline is substituted in, dot is left pointing at the last line created.

1.2.12 Joining Lines

Lines may also be joined together, but this is done with the **j** command instead of **s**. Given the lines

```
Now is  
□the time
```

and supposing that dot is set to the first of them, then the command

```
j
```

joins them together. No blanks are added, which is why we carefully showed a blank at the beginning of the second line.

All by itself, a **j** command joins line dot to line dot + 1, but any contiguous set of lines can be joined. Just specify the starting and ending line numbers. For example,

```
1,$jp
```

joins all the lines into one big one and prints it.

1.2.13 Rearranging a Line with \(...\)

Recall that ‘&’ is a shorthand that stands for whatever was matched by the left side of an **s** command. In much the same way you can capture separate pieces of what was matched; the only difference is that you have to specify on the left side just what pieces you’re interested in.

ADVANCED EDITING

Suppose, for instance, that you have a file of lines that consist of names in the form

Smith, A. B.
Jones, C.

and so on, and you want the initials to precede the name, as in

A. B. Smith
C. Jones

It is possible to do this with a series of editing commands, but it is tedious and error-prone. (It is instructive to figure out how it is done, though.)

The alternative is to ‘tag’ the pieces of the pattern (in this case, the last name, and the initials), and then rearrange the pieces. On the left side of a substitution, if part of the pattern is enclosed between `\(`` and `\)``, whatever matched that part is remembered, and available for use on the right side. On the right side, the symbol ‘`\1`’ refers to whatever matched the first `\(...\)` pair, ‘`\2`’ to the second `\(...\)`, and so on.

The command

1,\$\$/^\\([^,]*\\),*\\(.*)/\\2\\1/

although hard to read, does the job. The first `\(...\)` matches the last name, which is any string up to the comma; this is referred to on the right side with ‘`\1`’. The second `\(...\)` is whatever follows the comma and any spaces, and is referred to as ‘`\2`’.

Of course, with any editing sequence this complicated, it’s foolhardy to simply run it and hope. The global commands `g` and `v` provide a way for you to print exactly those lines which were affected by the substitute command, and thus verify that it did what you wanted in all cases.

1.3 LINE ADDRESSING IN THE EDITOR

The next general area we will discuss is that of line addressing in **ed**, that is, how you specify what lines are to be affected by editing commands. We have already used constructions like

1,\$s/x/y/

to specify a change on all lines. And most users are long since familiar with using a single newline (or return) to print the next line, and with

/thing/

to find a line that contains ‘thing’. Less familiar, surprisingly enough, is the use of

?thing?

to scan backwards for the previous occurrence of ‘thing’. This is especially handy when you realize that the thing you want to operate on is back up the page from where you are currently editing.

The slash and question mark are the only characters you can use to delimit a context search, though you can use essentially any character in a substitute command.

1.3.1 Address Arithmetic

The next step is to combine the line numbers like ‘.’, ‘\$’, ‘/.../’ and ‘?...?’ with ‘+’ and ‘-’. Thus

\$ -1

is a command to print the next to last line of the current file (that is, one line before line ‘\$’). For example, to recall how far you got in a previous editing session,

\$ -5,\$p

prints the last six lines. (Be sure you understand why it’s six, not five.) If there aren’t six, of course, you’ll get an error message.

ADVANCED EDITING

As another example,

.-3,.+3p

prints from three lines before where you are now (at line dot) to three lines after, thus giving you a bit of context. By the way, the '+' can be omitted:

.-3,.3p

is absolutely identical in meaning.

Another area in which you can save typing effort in specifying lines is to use '-' and '+' as line numbers by themselves.

-

by itself is a command to move back up one line in the file. In fact, you can string several minus signs together to move back up that many lines:

moves up three lines, as does '-3'. Thus

-3,+3p

is also identical to the examples above.

Since '-' is shorter than '.-1', constructions like

-,.s/bad/good/

are useful. This changes 'bad' to 'good' on the previous line and on the current line.

'+' and '-' can be used in combination with searches using '/.../' and '?...?', and with '\$'. The search

/thing/- -

finds the line containing 'thing', and positions you two lines before it.

1.3.2 Repeated Searches

Suppose you ask for the search

/horrible thing/

and when the line is printed you discover that it isn't the horrible thing that you wanted, so it is necessary to repeat the search again. You don't have to re-type the search, for the construction

//

is a shorthand for 'the previous thing that was searched for', whatever it was. This can be repeated as many times as necessary. You can also go backwards:

??

searches for the same thing, but in the reverse direction.

Not only can you repeat the search, but you can use '**//**' as the left side of a substitute command, to mean 'the most recent pattern'.

/horrible thing/

.... ed prints line with 'horrible thing' ...

s//good/p

To go backwards and change a line, say

??s//good/

Of course, you can still use the '&' on the right hand side of a substitute to stand for whatever got matched:

//s//&□&/p

finds the next occurrence of whatever you searched for last, replaces it by two copies of itself, then prints the line just to verify that it worked.

ADVANCED EDITING

1.3.3 Default Line Numbers and the Value of Dot

One of the most effective ways to speed up your editing is always to know what lines will be affected by a command if you don't specify the lines it is to act on, and on what line you will be positioned (i.e., the value of dot) when a command finishes. If you can edit without specifying unnecessary line numbers, you can save a lot of typing.

As the most obvious example, if you issue a search command like

/thing/

you are left pointing at the next line that contains 'thing'. Then no address is required with commands like **s** to make a substitution on that line, or **p** to print it, or **l** to list it, or **d** to delete it, or **a** to append text after it, or **c** to change it, or **i** to insert text before it.

What happens if there was no 'thing'? Then you are left right where you were — dot is unchanged. This is also true if you were sitting on the only 'thing' when you issued the command. The same rules hold for searches that use '?...?'; the only difference is the direction in which you search.

The delete command **d** leaves dot pointing at the line that followed the last deleted line. When line '\$' gets deleted, however, dot points at the new line '\$'.

The line-changing commands **a**, **c** and **i** by default all affect the current line, if you give no line number with them, **a** appends text after the current line, **c** changes the current line, and **i** inserts text before the current line.

a, **c**, and **i** behave identically in one respect. When you stop appending, changing or inserting, dot points at the last line entered. This is exactly what you want for typing and editing on the fly. For example, you can say

```

a
... text ...
... botch ...      (minor error)

.
s/botch/correct/   (fix botched line)
a
... more text ...

```

without specifying any line number for the substitute command or for the second append command. Or you can say

```

a
... text ...
... horrible botch ... (major error)

.
c          (replace entire line)
... fixed up line ...

```

You should experiment to determine what happens if you add no lines with **a**, **c** or **i**.

The **r** command will read a file into the text being edited, either at the end if you give no address, or after the specified line if you do. In either case, dot points at the last line read in. Remember that you can even say **0r** to read a file in at the beginning of the text. (You can also say **0a** or **1i** to start adding text at the beginning.)

The **w** command writes out the entire file. If you precede the command by one line number, that line is written, while if you precede it by two line numbers, that range of lines is written. The **w** command does not change dot: the current line remains the same, regardless of what lines are written. This is true even if you say something like

```
/^\^.AB/,^\^.AE/w abstract
```

which involves a context search.

ADVANCED EDITING

Since the **w** command is so easy to use, you should save what you are editing regularly as you go along just in case the system crashes, or in case you do something foolish, like clobbering what you're editing.

The least intuitive behavior, in a sense, is that of the **s** command. The rule is simple — you are left sitting on the last line that got changed. If there were no changes, then dot is unchanged.

To illustrate, suppose that there are three lines in the buffer, and you are sitting on the middle one:

```
x1  
x2  
x3
```

Then the command

```
- , + s/x/y/p
```

prints the third line, which is the last one changed. But if the three lines had been

```
x1  
y2  
y3
```

and the same command had been issued while dot pointed at the second line, then the result would be to change and print only the first line, and that is where dot would be set.

1.3.4 Semicolon ‘;’

Searches with ‘/.../’ and ‘?...?’ start at the current line and move forward or backward respectively until they either find the pattern or get back to the current line. Sometimes this is not what is wanted. Suppose, for example, that the buffer contains lines like this:

```
.  
. .  
ab  
. .  
. .  
bc  
. .
```

Starting at line 1, one would expect that the command

```
/a/,/b/p
```

prints all the lines from the ‘ab’ to the ‘bc’ inclusive. Actually this is not what happens. Both searches (for ‘a’ and for ‘b’) start from the same point, and thus they both find the line that contains ‘ab’. The result is to print a single line. Worse, if there had been a line with a ‘b’ in it before the ‘ab’ line, then the print command would be in error, since the second line number would be less than the first, and it is illegal to try to print lines in reverse order.

This is because the comma separator for line numbers doesn’t set dot as each address is processed; each search starts from the same place. In **ed**, the semicolon ‘;’ can be used just like comma, with the single difference that use of a semicolon forces dot to be set at that point as the line numbers are being evaluated. In effect, the semicolon ‘moves’ dot. Thus in our example above, the command

```
/a;/b/p
```

prints the range of lines from ‘ab’ to ‘bc’, because after the ‘a’ is found, dot is set to that line, and then ‘b’ is searched for, starting beyond that line.

This property is most often useful in a very simple situation. Suppose you want to find the second occurrence of ‘thing’. You could say

```
/thing/  
//
```

ADVANCED EDITING

but this prints the first occurrence as well as the second, and is a nuisance when you know very well that it is only the second one you're interested in. The solution is to say

/thing/;//

This says to find the first occurrence of 'thing', set dot to that line, then find the second and print only that.

Closely related is searching for the second previous occurrence of something, as in

?something?;??

Printing the third or fourth or ... in either direction is left as an exercise.

Finally, bear in mind that if you want to find the first occurrence of something in a file, starting at an arbitrary place within the file, it is not sufficient to say

1;/thing/

because this fails if 'thing' occurs on line 1. But it is possible to say

0;/thing/

(one of the few places where 0 is a legal line number), for this starts the search at line 1.

1.3.5 Interrupting the Editor

As a final note on what dot gets set to, you should be aware that if you hit the interrupt or '^C' or break key while **ed** is doing a command, things are put back together again and your state is restored as much as possible to what it was before the command began. Naturally, some changes are irrevocable. If you are reading or writing a file or making substitutions or deleting lines, these will be stopped in some clean but unpredictable state in the middle (which is why it is not usually wise to stop them). Dot may or may not be changed.

Printing is more clear cut. Dot is not changed until the printing is done. Thus if you print until you see an interesting line, then hit ‘~C’, you are not sitting on that line or even near it. Dot is left where it was when the **p** command was started.

1.4 GLOBAL COMMANDS

The global commands **g** and **v** are used to perform one or more editing commands on all lines that either contain (**g**) or don’t contain (**v**) a specified pattern.

As the simplest example, the command

g/VENIX/p

prints all lines that contain the word ‘VENIX’. The pattern that goes between the slashes can be anything that could be used in a line search or in a substitute command; exactly the same rules and limitations apply.

As another example, then,

g/^./p

prints all the formatting commands in a file (lines that begin with ‘.’).

The **v** command is identical to **g**, except that it operates on those line that do not contain an occurrence of the pattern. (Don’t look too hard for mnemonic significance to the letter ‘v’.) So

v/^./p

prints all the lines that don’t begin with ‘.’ — the actual text lines.

The command that follows **g** or **v** can be anything:

g/^./d

deletes all lines that begin with ‘.’, and

ADVANCED EDITING

g/^\$/d

deletes all empty lines.

Probably the most useful command that can follow a global is the substitute command, for this can be used to make a change and print each affected line for verification. For example, we could change the word ‘Venix’ to ‘VENIX’ everywhere, and verify that it really worked, with

g/Venix/s//VENIX/gp

Notice that we used ‘//’ in the substitute command to mean ‘the previous pattern’, in this case, ‘Venix’. The **p** command is done on every line that matches the pattern, not just those on which a substitution took place.

The global command operates by making two passes over the file. On the first pass, all lines that match the pattern are marked. On the second pass, each marked line in turn is examined, dot is set to that line, and the command executed. This means that it is possible for the command that follows a **g** or **v** to use addresses, set dot, and so on, quite freely.

g/^\.PP/+

prints the line that follows each ‘.PP’ command (the signal for a new paragraph in some formatting packages). Remember that ‘+’ means ‘one line past dot’. And

g/topic/?^\.SH?1

searches for each line that contains ‘topic’, scans backwards until it finds a line that begins ‘.SH’ (a section heading) and prints the line that follows that, thus showing the section headings under which ‘topic’ is mentioned. Finally,

g/^\.EQ/+ ,/^\.EN/-p

prints all the lines that lie between lines beginning with ‘.EQ’ and ‘.EN’ formatting commands.

The **g** and **v** commands can also be preceded by line numbers, in which case the lines searched are only those in the range specified.

1.4.1 Multi-line Global Commands

It is possible to do more than one command under the control of a global command, although the syntax for expressing the operation is not especially natural or pleasant. As an example, suppose the task is to change ‘x’ to ‘y’ and ‘a’ to ‘b’ on all lines that contain ‘thing’. Then

```
g/thing/s/x/y/\
s/a/b/
```

is sufficient. The ‘\’ signals the **g** command that the set of commands continues on the next line; it terminates on the first line that does not end with ‘\’. (As a minor blemish, you can’t use a substitute command to insert a newline within a **g** command.)

You should watch out for this problem: the command

```
g/x/s//y/\
s/a/b/
```

does not work as you expect. The remembered pattern is the last pattern that was actually executed, so sometimes it will be ‘x’ (as expected), and sometimes it will be ‘a’ (not expected). You must spell it out, like this:

```
g/x/s/x/y/\
s/a/b/
```

It is also possible to execute **a**, **c** and **i** commands under a global command; as with other multi-line constructions, all that is needed is to add a ‘\’ at the end of each line except the last. Thus to add a ‘.nf’ and ‘.sp’ command before each ‘.EQ’ line, type

```
g/^\.EQ/i\
.nf\
.sp
```

ADVANCED EDITING

There is no need for a final line containing a ‘.’ to terminate the **i** command, unless there are further commands being done under the global. On the other hand, it does no harm to put it in either.

1.5 CUT AND PASTE WITH VENIX COMMANDS

One editing area which is very useful is ‘cut and paste’ operations — changing the name of a file, making a copy of a file somewhere else, moving a few lines from one place to another in a file, inserting one file in the middle of another, splitting a file into pieces, and splicing two or more files together.

The next several sections talk about cut and paste. We will begin with the VENIX commands for moving entire files around, then discuss **ed** commands for operating on pieces of files.

1.5.1 Changing the Name of a File

You have a file named ‘memo’ and you want it to be called ‘paper’ instead. How is it done?

The VENIX program that renames files is called **mv** (for ‘move’); it ‘moves’ the file from one name to another, like this:

mv memo paper

That’s all there is to it: **mv** from the old name to the new name.

mv oldname newname

Warning: if there is already a file around with the new name, its present contents will be silently clobbered by the information from the other file. The one exception is that you can’t move a file to itself —

mv x x

is illegal.

1.5.2 Making a Copy of a File

Sometimes what you want is a copy of a file. This might be because you want to work on a file, and yet save a copy in case something gets fouled up.

The way to make a copy is with the **cp** command. (**cp** stands for ‘copy’.) Suppose you have a file called ‘good’ and you want to save a copy before you make some dramatic editing changes. Choose a name — for instance ‘**savegood**’ — then type

```
cp good savegood
```

This copies ‘good’ onto ‘**savegood**’, and you now have two identical copies of the file ‘good’. (If ‘**savegood**’ previously contained something, it gets overwritten.)

Now if you decide at some time that you want to get back to the original state of ‘good’, you can say

```
mv savegood good
```

(if you’re not interested in ‘**savegood**’ any more), or

```
cp savegood good
```

if you still want to retain a safe copy.

In summary, **mv** just renames a file; **cp** makes a duplicate copy. Both of them clobber the ‘target’ file if it already exists, so you had better be sure that’s what you want to do before you do it.

1.5.3 Removing a File

If you decide you are really done with a file forever, you can remove it with the **rm** command:

```
rm savegood
```

throws away (irrevocably) the file called ‘**savegood**’.

ADVANCED EDITING

1.5.4 Putting Two or More Files Together

The next step is the familiar one of collecting two or more files into one big one. This will be needed, for example, when the author decides that several sections need to be combined into one. There are several ways to do it, of which the cleanest is a program called **cat**. **cat** is short for ‘concatenate’, which is exactly what we want to do.

Suppose the job is to combine the files ‘file1’ and ‘file2’ into a single file called ‘bigfile’. If you say

```
cat file
```

the contents of ‘file’ will get printed on your terminal. If you say

```
cat file1 file2
```

the contents of ‘file1’ and then the contents of ‘file2’ will both be printed on your terminal, in that order.

cat combines the files, but it’s not much help to print them on the terminal — we want them in ‘bigfile’. Fortunately, there is a way. You can tell the system that instead of printing on your terminal, you want the same information put in a file. The way to do it is to add to the command line the character > and the name of the file where you want the output to go. Then you can say

```
cat file1 file2 >bigfile
```

and the job is done. (As with **cp** and **mv**, you’re putting something into ‘bigfile’, and anything that was already there is destroyed.)

This ability to ‘capture’ the output of a program is one of the most useful aspects of the VENIX system. Fortunately it’s not limited to the **cat** program — you can use it with any program that prints on your terminal. We’ll see some more uses for it in a moment.

Naturally, you can combine several files, not just two:

```
cat file1 file2 file3 ... >bigfile
```

collects a whole bunch.

Question: is there any difference between

```
cp good savegood
```

and

```
cat good >savegood
```

Answer: for most purposes, no. You might reasonably ask why there are two programs in that case, since **cat** is obviously all you need. The answer is that **cp** will do some other things as well, as documented in the *User Reference Manual*. For now we'll stick to simple usages.

1.5.5 Adding Something to the End of a File

Sometimes you want to add one file to the end of another. We have enough building blocks now to do it. We can use **cp**, **mv** and **cat** to add the file 'good1' to the end of the file 'good'.

One method is

```
cat good good1 >temp  
mv temp good
```

which is probably most direct. You should also understand why

```
cat good good1 >good
```

doesn't work. (Don't practice with a good 'good'!)

The easy way is to use a variant of **>**, called **>>**. In fact, **>>** is identical to **>** except that instead of overwriting the old file, **>>** simply appends the new material at the end of the old file. Thus you could say

```
cat good1 >>good
```

in order to add 'good1' to the end of 'good'. (And if 'good' didn't exist, this makes a copy of 'good1' called 'good'.)

ADVANCED EDITING

1.6 CUT AND PASTE WITH THE EDITOR

Now we move on to manipulating pieces of files, individual lines or groups of lines.

1.6.1 Filenames

The first step is to ensure that you know the **ed** commands for reading and writing files. Of course you can't go very far without knowing **r** and **w**. Equally useful, but less well known, is the 'edit' command **e**. Within **ed**, the command

```
e newfile
```

says 'I want to edit a new file called *newfile*, without leaving the editor.' The **e** command discards whatever you're currently working on and starts over on *newfile*. It's exactly the same as if you had quit with the **q** command, then re-entered **ed** with a new file name, except that if you have a pattern remembered, then a command like **//** will still work.

If you enter **ed** with the command

```
ed file
```

ed remembers the name of the file, and any subsequent **e**, **r** or **w** commands that don't contain a filename will refer to this remembered file. Thus

```
ed file1
... (editing) ...
w      (writes back in file1)
e  file2 (edit new file, without leaving editor)
... (editing on file2) ...
w      (writes back on file2)
```

(and so on) does a series of edits on various files without ever leaving **ed** and without typing the name of any file more than once.

You can find out the remembered file name at any time with the **f** command; just type **f** without a file name. You can also change the name of the remembered file name with **f**; a useful sequence is

```
ed  precious
f  junk
... (editing) ...
```

which gets a copy of a *precious*, then uses **f** to guarantee that a careless **w** command won't overwrite the original.

1.6.2 Inserting One File into Another

Suppose you have a file called 'memo', and you want the file called 'table' to be inserted just after the reference to Table 1. That is, in 'memo' somewhere is a line that says

Table 1 shows that ...

and the data contained in 'table' has to go there, probably so it will be formatted properly by **nroff**. Now what?

This one is easy. Edit 'memo', find 'Table 1', and add the file 'table' right there:

```
ed  memo
/Table 1/
Table 1 shows that ... [response from ed]
.r  table
```

The critical line is the last one. As we said earlier, the **r** command reads a file; here you asked for it to be read in right after line dot. An **r** command without any address adds lines at the end, so it is the same as **\$r**.

1.6.3 Writing Out Part of a File

The other side of the coin is writing out part of the document you're editing. For example, maybe you want to put into a separate file that table from the previous example, so it can be formatted and tested separately. Suppose that in the file being edited we have

```
.TS
...[lots of stuff]
.TE
```

ADVANCED EDITING

which is the way a table is set up for the **tbl** program. To isolate the table in a separate file called ‘table’, first find the start of the table (the ‘.TS’ line), then write out the interesting part:

```
/^\.TS/  
.TS [ed prints the line it found]  
.,"/^\.TE/w table
```

and the job is done. If you are confident, you can do it all at once with

```
/^\.TS/;/^\.TE/w table
```

The **w** command can write out a group of lines, instead of the whole file. In fact, you can write out a single line if you like; just give one line number instead of two. For example, if you have just typed a complicated line and you know that it (or something like it) is going to be needed later, then save it, don’t re-type it. An example of this feature is

```
a  
...lots of stuff...  
...horrible line...  
  
.w temp  
a  
...more stuff...  
  
.r temp  
a  
...more stuff...  
.
```

1.6.4 Rearranging Test

Suppose you want to move a paragraph from its present position in a paper to the end. How would you do it?

Let's assume that each paragraph in the paper begins with the formatting command '.PP'. One method is to write the paragraph onto a temporary file, delete it from its current position, then read in the temporary file at the end. Assuming that the editor is sitting on the '.PP' command that begins the paragraph, this is the sequence of commands:

```
.,/^\.PP/-w temp
.,//d
$ r temp
```

That is, from where you are now ('.') until one line before the next '.PP' ('/^\.PP/-') write onto 'temp'. Then delete the same lines. Finally, read 'temp' at the end.

Often, an easier way is to use the move command **m** that **ed** provides. It lets you do the whole set of operations at one crack, without any temporary file.

The **m** command is like many other **ed** commands in that it takes up to two line numbers in front that tell which lines are to be affected. It is also followed by a line number that tells where the lines are to go. Thus

```
line1, line2 m line3
```

says to move all the lines between 'line1' and 'line2' after 'line3'. Naturally, any of 'line1' etc., can be patterns between slashes, \$ signs, or other ways to specify lines.

Suppose again that the editor is sitting at the first line of the paragraph. Then you can say

```
.,/^\.PP/-m$
```

That's all that's needed.

As another example of a frequent operation, you can reverse the order of two adjacent lines by moving the first one to the end of the second. Suppose that you are positioned at the first. Then

ADVANCED EDITING

m +

accomplishes this switch. It says to move line dot to the end of the one line following line dot. If you are positioned on the second line,

m --

does the interchange.

As you can see, the **m** command is more succinct and direct than writing, deleting and re-reading. The main difficulty with the **m** command is that when using patterns to specify both the lines you are moving and the target, you must specify them properly, or you may move the wrong lines accidentally. The result of a botched **m** command can be a ghastly mess. Doing the job a step at a time makes it easier for you to verify at each step that you accomplished what you wanted to. It's also a good idea to issue a **w** command before doing anything complicated; then if you make a mistake, it's easy to back up to where you were before the mistake was made.

1.6.5 Marks

ed provides a facility for marking a line with a particular name so you can later reference it by name regardless of its actual line number. This can be handy for moving lines, and for keeping track of them as they move. The mark command is **k**; the command

kx

marks the current line with the name 'x'. A line elsewhere in the file can be marked by preceding **k** with a line number. (The mark name must be a single lower case letter.) Now you can refer to the marked line with the address

'x

Marks are useful for moving text around. Find the first line of the block to be moved, and mark it with '**a**'. Then find the last line and mark it with '**b**'. Now place the editor where the block is to go and say

'a,'bm.

Bear in mind that only one line can have a particular mark name associated with it at any given time.

1.6.6 Copying Lines

We mentioned earlier the idea of saving a line that was hard to type or used often, so as to cut down on typing time.

ed provides another command, called **t** (for ‘transfer’) for making a copy of a group of one or more lines at any point. This is often easier than writing and reading.

The **t** command is identical to the **m** command, except that instead of moving lines it simply duplicates them at the place you named. Thus

1,\$t\$

duplicates the entire contents that you are editing. A more common use for **t** is for creating a series of lines that differ only slightly. For example, you can say

```
a
..... x ..... (long line)
.
t.          (make a copy)
s/x/y/      (change it a bit)
t.          (make third copy)
s/y/z/      (change it a bit)
```

and so on.

1.6.7 The Temporary Escape ‘!’

Sometimes it is convenient to be able to temporarily escape from the editor to do some other VENIX command without actually leaving the editor. The ‘escape’ command **!** provides a way to do this.

If you say

!any VENIX command

your current editing state is suspended, and the VENIX command you asked for is executed. When the command finishes, **ed** will signal you by printing another **!**; at that point you can resume editing.

You can really do any VENIX command, including another **ed**. (This is quite common, in fact.) In this case, you can even do another **!**.

ADVANCED EDITING

1.7 SUPPORTING TOOLS

There are several tools and techniques that go along with the editor, all of which are relatively easy once you know how **ed** works, because they are all based on the editor. In this section we give some cursory examples of these tools, more to indicate their existence than to provide a complete tutorial. More information on each can be found in the *User Reference Manual*.

1.7.1 Grep

Sometimes you want to find all occurrences of some word or pattern in a set of files, to edit them or perhaps just to verify their presence or absence. It may be possible to edit each file separately and look for the pattern of interest. In cases where there are many files, this can get very tedious, and if the files are very big, it may be impossible because of limits in **ed**.

The program **grep** was invented to get around these limitations. The search patterns that we have described in the paper are often called ‘regular expressions’, and ‘grep’ stands for

g/re/p

grep prints every line in a set of files that contains a particular pattern. Thus

grep 'thing' file1 file2 file3 ...

finds ‘thing’ wherever it occurs in any of the files ‘file1’, ‘file2’, etc. **grep** also indicates the file in which the line was found, so you can later edit it if you like.

The pattern represented by ‘thing’ can be any pattern you can use in the editor, since **grep** and **ed** use exactly the same mechanism for pattern searching. It is wisest always to enclose the pattern in the single quotes ‘...’ if it contains any non-alphabetic characters, since many such characters also mean something special to the VENIX command interpreter (the ‘shell’). If you don’t quote them, the command interpreter will try to interpret them before **grep** gets a chance.

There is also a way to find lines that don’t contain a pattern:

grep -v 'thing' file1 file2 ...

finds all lines that don’t contain ‘thing’. The **-v** must occur in the position shown. Given **grep** and **grep -v**, it is possible to do things like selecting all lines that contain some combination of patterns. For example, to get all lines

that contain ‘x’ but not ‘y’:

```
grep x file... | grep -v y
```

(The notation | is a ‘pipe’, which causes the output of the first command to be used as input to the second command; see VENIX FOR BEGINNERS in the *User Guide*.)

1.7.2 Editing Scripts

If a fairly complicated set of editing operations is to be done on a whole set of files, the easiest thing to do is to make up a ‘script’, i.e., a file that contains the operations you want to perform, then apply this script to each file in turn.

For example, suppose you want to change every ‘Venix’ to ‘VENIX’ and every ‘Unix’ to ‘UNIX’ in a large number of files. Then put into the file ‘script’ the lines

```
g/Venix/s//VENIX/g
g/Unix/s//UNIX/g
w
q
```

Now you can say

```
ed file1 <script
ed file2 <script
...

```

This causes **ed** to take its commands from the prepared script. Notice that the whole job has to be planned in advance.

And of course by using the VENIX command interpreter, you can cycle through a set of files automatically, with varying degrees of ease.

1.7.3 Sed

sed (‘stream editor’) is a version of the editor with restricted capabilities but which is capable of processing unlimited amounts of input. Basically, **sed** copies its input to its output, applying one or more editing commands to each line of input.

ADVANCED EDITING

As an example, suppose that we want change every ‘Venix’ to ‘VENIX’, as was done above, but without rewriting the files. Then the command

```
sed 's/Venix/VENIX/g' file1 file2 ...
```

applies the command ‘s/Venix/VENIX/g’ to all lines from ‘file1’, ‘file2’, etc., and copies all lines to the output. The advantage of using **sed** in such a case is that it can be used with input too large for **ed** to handle. All the output can be collected in one place, either in a file or perhaps piped into another program.

If the editing transformation is so complicated that more than one editing command is needed, commands can be supplied from a file, or on the command line, with a slightly more complex syntax. To take commands from a file, for example,

```
sed -f cmdfile input-files...
```

sed has further capabilities, including conditional testing and branching, which we cannot go into here; (see **SED — A NON-INTERACTIVE TEXT EDITOR**, chapter 2).

CONTENTS

2

2.1 INTRODUCTION	2-1
2.2 OVERALL OPERATION	2-2
2.3 ADDRESSES: SELECTING LINES FOR EDITING	2-4
2.4 FUNCTIONS	2-6

Chapter 2

SED — A NON-INTERACTIVE TEXT EDITOR

2.1 INTRODUCTION

sed is a non-interactive context editor designed to be especially useful in three cases:

- 1) To edit files too large for comfortable interactive editing;
- 2) To edit any size file when the sequence of editing commands is too complicated to be comfortably typed in interactive mode;
- 3) To perform multiple ‘global’ editing functions efficiently in one pass through the input.

Since only a few lines of the input reside in core at one time, and no temporary files are used, the effective size of file that can be edited is limited only by the requirement that the input and output fit simultaneously into available secondary storage.

Complicated editing scripts can be created separately and given to **sed** as a command file. For complex edits, this saves considerable typing, and its attendant errors. **sed** running from a command file is much more efficient than other interactive editors, even if that editor can be driven by a pre-written script.

The principal loss of functions compared to an interactive editor are lack of relative addressing (because of the line-at-a-time operation), and lack of immediate verification that a command has done what was intended.

SED

sed is a lineal descendant of the VENIX editor, **ed**. Because of the differences between interactive and non-interactive operation, considerable changes have been made between **ed** and **sed**. The most striking family resemblance between the two editors is in the class of patterns ('regular expressions') they recognize.

2.2 OVERALL OPERATION

sed by default copies the standard input to the standard output, perhaps performing one or more editing commands on each line before writing it to the output. This behavior may be modified by flags on the command line.

The general format of an editing command is:

[address1,address2][function][arguments]

One or both addresses may be omitted; the format of addresses is given in the section "Addresses." Any number of blanks or tabs may separate the addresses from the function. The function must be present; the available commands are discussed in the section "Functions." The arguments may be required or optional, according to which function is given; again, they are discussed in "Functions" under each individual function.

Tab characters and spaces at the beginning of lines are ignored.

2.2.1 Command-line Flags

Three flags are recognized on the command line:

- n:** tells **sed** not to copy all lines, but only those specified by **p** functions or **p** flags after **s** functions;
- e:** tells **sed** to take the next argument as an editing command;
- f:** tells **sed** to take the next argument as a file name; the file should contain editing commands, one to a line.

2.2.2 Order of Application of Editing Commands

Before any editing is done (in fact, before any input file is even opened), all the editing commands are compiled into a form which will be moderately efficient during the execution phase (when the commands are actually applied to lines of the input file). The commands are compiled in the order in which they are encountered; this is generally the order in which they will be attempted at execution time. The commands are applied one at a time; the input to each command is the output of all preceding commands.

The default linear order of application of editing commands can be changed by the flow-of-control commands, **t** and **b**. Even when the order of application is changed by these commands, it is still true that the input line to any command is the output of any previously applied command.

2.2.3 Pattern-space

The range of pattern matches is called the pattern space. Ordinarily, the pattern space is one line of the input text, but more than one line can be read into the pattern space by using the **N** command.

2.2.4 Examples

Examples are scattered throughout the text. Except where otherwise noted, the examples all assume the following input text:

**In Xanadu did Kubla Khan
A stately pleasure dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.**

(In no case is the output of the **sed** commands to be considered an improvement on Coleridge.)

Example:

The command

2q

SED

will quit after copying the first two lines of the input. The output will be:

In Xanadu did Kubla Khan
A stately pleasure dome decree:

2.3 ADDRESSES: SELECTING LINES FOR EDITING

Lines in the input file(s) to which editing commands are to be applied can be selected by addresses. Addresses may be either line numbers or context addresses.

The application of a group of commands can be controlled by one address (or address-pair) by grouping the commands with curly braces ('{ }').

2.3.1 Line-number Addresses

A line number is a decimal integer. As each line is read from the input, a line-number counter is incremented; a line-number address matches (selects) the input line which causes the internal counter to equal the address line-number. The counter runs cumulatively through multiple input files; it is not reset when a new input file is opened.

As a special case, the character \$ matches the last line of the last input file.

2.3.2 Context Addresses

A context address is a pattern ('regular expression') enclosed in slashes ('/'). The regular expressions recognized by sed are constructed as follows:

- 1) An ordinary character (not one of those discussed below) is a regular expression, and matches that character.
- 2) A circumflex '^' at the beginning of a regular expression matches the null character at the beginning of a line.
- 3) A dollar-sign '\$' at the end of a regular expression matches the null character at the end of a line.

- 4) The characters ‘\n’ match an imbedded newline character, but not the newline at the end of the pattern space.
- 5) A period ‘.’ matches any character except the terminal newline of the pattern space.
- 6) A regular expression followed by an asterisk “*” matches any number (including 0) of adjacent occurrences of the regular expression it follows.
- 7) A string of characters in square brackets ‘[]’ matches any character in the string, and no others. If, however, the first character of the string is circumflex “^”, the regular expression matches any character **except** the characters in the string and the terminal newline of the pattern space.
- 8) A concatenation of regular expressions is a regular expression which matches the concatenation of strings matched by the components of the regular expression.
- 9) A regular expression between the sequences ‘\(` and ‘\)`’ is identical in effect to the unadorned regular expression, but has side-effects which are described under the **s** command below and specification 10) immediately below.
- 10) The expression ‘\d’ means the same string of characters matched by an expression enclosed in ‘\(` and ‘\)`’ earlier in the same pattern. Here **d** is a single digit; the string specified is that beginning with the *d*th occurrence of ‘\(` counting from the left. For example, the expression ‘\(.*\)\1’ matches a line beginning with two repeated occurrences of the same string.
- 11) The null regular expression standing alone (e.g., ‘//’) is equivalent to the last regular expression compiled.

To use one of the special characters (‘\$’ . * [] \ /) as a literal (to match an occurrence of itself in the input), precede the special character by a backslash ‘\’.

For a context address to ‘match’ the input requires that the whole pattern within the address match some portion of the pattern space.

SED

2.3.3 Number of Addresses

The commands in the next section can have 0, 1, or 2 addresses. Under each command the maximum number of allowed addresses is given. For a command to have more addresses than the maximum allowed is considered an error.

If a command has no addresses, it is applied to every line in the input.

If a command has one address, it is applied to all lines which match that address.

If a command has two addresses, it is applied to the first line which matches the first address, and to all subsequent lines until (and including) the first subsequent line which matches the second address. Then an attempt is made on subsequent lines to again match the first address, and the process is repeated.

Two addresses are separated by a comma.

Examples:

/an/	matches lines 1, 3, 4 in our sample text
/an.*an/	matches line 1
/an/	matches no lines
	matches all lines
/^. /	matches line 5
/r*an/	matches lines 1,3, 4 (number = zero!)
/(an\).*\1/	matches line 1

2.4 FUNCTIONS

All functions are named by a single character. In the following summary, the maximum number of allowable addresses is given enclosed in parentheses, then the single character function name, possible arguments enclosed in angles (< >), an expanded English translation of the single-character name, and finally a description of what each function does. The angles around the arguments are not part of the argument, and should not be typed in actual editing commands.

2.4.1 Whole-line Oriented Functions

(2)d —

delete lines

The **d** function deletes from the file (does not write to the output) all those lines matched by its address(es).

It also has the side effect that no further commands are attempted on the corpse of a deleted line; as soon as the **d** function is executed, a new line is read from the input, and the list of editing commands is re-started from the beginning on the new line.

(2)n —

next line

The **n** function reads the next line from the input, replacing the current line. The current line is written to the output if it should be. The list of editing commands is continued following the **n** command.

(1)a\

<text> —

append lines

The **a** function causes the argument <text> to be written to the output after the line matched by its address. The **a** command is inherently multi-line; **a** must appear at the end of a line, and <text> may contain any number of lines. To preserve the one-command-to-a-line fiction, the interior newlines must be hidden by a backslash character ('\') immediately preceding the newline. The <text> argument is terminated by the first unhidden newline (the first one not immediately preceded by backslash).

Once an **a** function is successfully executed, <text> will be written to the output regardless of what later commands do to the line which triggered it. The triggering line may be deleted entirely; <text> will still be written to the output.

The <text> is not scanned for address matches, and no editing commands are attempted on it. It does not cause any change in the line-number counter.

SED

(1)i\

<text> —

insert lines

The **i** function behaves identically to the **a** function, except that <text> is written to the output before the matched line. All other comments about the **a** function apply to the **i** function as well.

(2)c\

<text> —

change lines

The **c** function deletes the lines selected by its address(es), and replaces them with the lines in <text>. Like **a** and **i**, **c** must be followed by a newline hidden by a backslash; and interior new lines in <text> must be hidden by backslashes.

The **c** command may have two addresses, and therefore select a range of lines. If it does, all the lines in the range are deleted, but only one copy of <text> is written to the output, not one copy per line deleted. As with **a** and **i**, <text> is not scanned for address matches, and no editing commands are attempted on it. It does not change the line-number counter.

After a line has been deleted by a **c** function, no further commands are attempted on the corpse.

If text is appended after a line by **a** or **r** functions, and the line is subsequently changed, the text inserted by the **c** function will be placed before the text of the **a** or **r** functions.

Note: Within the text put in the output by these functions, leading blanks and tabs will disappear, as always in sed commands. To get leading blanks and tabs into the output, precede the first desired blank or tab by a backslash; the backslash will not appear in the output.

Example:

The list of editing commands:

```
n
a\
XXXX
d
```

applied to our standard input, produces:

```
In Xanadu did Kubhla Khan
XXXX
Where Alph, the sacred river, ran
XXXX
Down to a sunless sea.
```

In this particular case, the same effect would be produced by either of the two following command lists:

```
n          n
i\        c\
XXXX  XXXX
d
```

2.4.2 Substitute Function

One very important function changes parts of lines selected by a context search within the line.

(2)s<pattern><replacement><flags> — substitute

The s function replaces part of a line (selected by <pattern>) with <replacement>. It can best be read:

Substitute for <pattern>, <replacement>

The <pattern> argument contains a pattern, exactly like the patterns in addresses. The only difference between <pattern> and a context address is that the context address must be delimited by slash ('/') characters; <pattern> may be delimited by any character other than space or newline.

By default, only the first string matched by <pattern> is replaced, but see the g flag below.

The <replacement> argument begins immediately after the second

SED

delimiting character of <pattern>, and must be followed immediately by another instance of the delimiting character. (Thus there are exactly three instances of the delimiting character.)

The <replacement> is not a pattern, and the characters which are special in patterns do not have special meaning in <replacement>. Instead, other characters are special:

- & is replaced by the string matched by <pattern>
- \d (where d is a single digit) is replaced by the dth substring matched by parts of <pattern> enclosed in '\(' and '\)'. If nested substrings occur in <pattern>, the dth is determined by counting opening delimiters ('\(').

As in patterns, special characters may be made literal by preceding them with backslash ('\').

The <flags> argument may contain the following flags:

- g — substitute <replacement> for all (non-overlapping) instances of <pattern> in the line. After a successful substitution, the scan for the next instance of <pattern> begins just after the end of the inserted characters; characters put into the line from <replacement> are not rescanned.
- p — print the line if a successful replacement was done. The p flag causes the line to be written to the output if and only if a substitution was actually made by the s function. Notice that if several s functions, each followed by a p flag, successfully substitute in the same input line, multiple copies of the line will be written to the output: one for each successful substitution.

w <filename> —

write the line to a file if a successful replacement was done. The w flag causes lines which are actually substituted by the s function to be written to a file named by <filename>. If <filename> exists before sed is run, it is overwritten; if not, it is created.

A single space must separate w and <filename>.

The possibilities of multiple, somewhat different copies of one input line being written are the same as for p.

A maximum of 10 different file names may be mentioned after **w** flags and **w** functions (see below), combined.

Examples:

The following command, applied to our standard input,

s/to/by/w changes

produces, on the standard output:

In Xanadu did Kubhla Khan
A stately pleasure dome decree:
Where Alph, the sacred river, ran
Through caverns measureless by man
Down by a sunless sea.

and, on the file ‘changes’:

Through caverns measureless by man
Down by a sunless sea.

If the nocopy option is in effect, the command:

s/[.,;?:]/*P&*/gp

produces:

A stately pleasure dome decree*P:*
Where Alph*P,* the sacred river*P,* ran
Down to a sunless sea*P.*

Finally, to illustrate the effect of the **g** flag, the command:

/X/s/an/AN/p

produces (assuming nocopy mode):

In XANadu did Kubhla Khan

and the command:

SED

/X/s/an/AN/gp

produces:

In XANadu did Kubhla KhAN

2.4.3 Input-output Functions

(2)p — print

The print function writes the addressed lines to the standard output file. They are written at the time the **p** function is encountered, regardless of what succeeding editing commands may do to the lines.

(2)w <filename> — write on <filename>

The write function writes the addressed lines to the file named by <filename>. If the file previously existed, it is overwritten; if not, it is created. The lines are written exactly as they exist when the write function is encountered for each line, regardless of what subsequent editing commands may do to them.

Exactly one space must separate the **w** and <filename>.

A maximum of ten different files may be mentioned in write functions and **w** flags after **s** functions, combined.

(1)r <filename> — read the contents of a file

The read function reads the contents of <filename>, and appends them after the line matched by the address. The file is read and appended regardless of what subsequent editing commands do to the line which matched its address. If **r** and **a** functions are executed on the same line, the text from the **a** functions and the **r** functions is written to the output in the order that the functions are executed.

Exactly one space must separate the **r** and <filename>. If a file mentioned by a **r** function cannot be opened, it is considered a null file, not an error, and no diagnostic is given.

NOTE: Since there is a limit to the number of files that can be opened simultaneously, care should be taken that no more than ten files be mentioned in **w** functions or flags; that number is reduced by one if any **r** functions are present. (Only one read file is open at one time.)

Examples:

Assume that the file ‘note1’ has the following contents:

**Note: Kubla Khan (more properly Kublai Khan; 1216-1294)
was the grandson and most eminent successor of Genghiz
(Chingiz) Khan, and founder of the Mongol dynasty in China.**

Then the following command:

/Kubla/r note1

produces:

In Xanadu did Kubla Khan

**Note: Kubla Khan (more properly Kublai Khan; 1216-1294)
was the grandson and most eminent successor of Genghiz
(Chingiz) Khan, and founder of the Mongol dynasty in China.**

**A stately pleasure dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.**

2.4.4 Input-line Functions

Three functions, all spelled with capital letters, deal specially with pattern spaces containing imbedded newlines; they are intended principally to provide pattern matches across lines in the input.

(2)N — Next line

The next input line is appended to the current line in the pattern space; the two input lines are separated by an imbedded newline. Pattern matches may extend across the imbedded newline(s).

(2)D — Delete first part of the pattern space

Delete up to and including the first newline character in the current pattern space. If the pattern space becomes empty (the only newline was the terminal newline), read another line from the input. In any case, begin the list of editing commands again from its beginning.

(2)P — Print first part of the pattern space

Print up to and including the first newline in the pattern space.

The **P** and **D** functions are equivalent to their lower-case counterparts if there are no imbedded newlines in the pattern space.

2.4.5 Hold and Get Functions

Four functions save and retrieve part of the input for possible later use.

(2)h — hold pattern space

The **h** functions copies the contents of the pattern space into a hold area (destroying the previous contents of the hold area).

(2)H — Hold pattern space

The **H** function appends the contents of the pattern space to the contents of the hold area; the former and new contents are separated by a newline.

(2)g — get contents of hold area

The **g** function copies the contents of the hold area into the pattern space (destroying the previous contents of the pattern space).

(2)G — Get contents of hold area

The **G** function appends the contents of the hold area to the contents of the pattern space; the former and new contents are separated by a newline.

(2)x — exchange

The exchange command interchanges the contents of the pattern space and the hold area.

Example:

The commands

```
1h
1s/ did.*//  

1x
G
s/>\n/ :/
```

applied to our standard example, produce:

```
In Xanadu did Kubla Khan :In Xanadu
A stately pleasure dome decree: :In Xanadu
Where Alph, the sacred river, ran :In Xanadu
Through caverns measureless to man :In Xanadu
Down to a sunless sea. :In Xanadu
```

2.4.6 Flow-of-Control Functions

These functions do no editing on the input lines, but control the application of functions to the lines selected by the address part.

(2)! — Don't

The **Don't** command causes the next command (written on the same line), to be applied to all and only those input lines **not** selected by the address part.

(2){ — Grouping

The grouping command '{' causes the next set of commands to be applied (or not applied) as a block to the input lines selected by the addresses of the grouping command. The first of the commands under control of the grouping may appear on the same line as the '{' or on the next line.

The group of commands is terminated by a matching '}' standing on a line by itself.

Groups can be nested.

SED

(0):<label> — place a label

The label function marks a place in the list of editing commands which may be referred to by **b** and **t** functions. The <label> may be any sequence of eight or fewer characters; if two different colon functions have identical labels, a compile time diagnostic will be generated, and no execution attempted.

(2)b<label> — branch to label

The branch function causes the sequence of editing commands being applied to the current input line to be restarted immediately after the place where a colon function with the same <label> was encountered. If no colon function with the same label can be found after all the editing commands have been compiled, a compile time diagnostic is produced, and no execution is attempted.

A **b** function with no <label> is taken to be a branch to the end of the list of editing commands; whatever should be done with the current input line is done, and another input line is read; the list of editing commands is restarted from the beginning on the new line.

(2)t<label> — test substitutions

The **t** function tests whether any successful substitutions have been made on the current input line; if so, it branches to <label>; if not, it does nothing. The flag which indicates that a successful substitution has been executed is reset by:

- 1) reading a new input line, or
- 2) executing a **t** function.

2.4.7 Miscellaneous Functions

(1)= — equals

The = function writes to the standard output the line number of the line matched by its address.

(1)q — quit

The **q** function causes the current line to be written to the output (if it should be), any appended or read text to be written, and execution to be terminated.

CONTENTS

3.1 INTRODUCTION	3-1
3.2 PARAGRAPHS	3-1
3.3 COVER SHEETS AND FIRST PAGES	3-2
3.4 PAGE HEADINGS	3-2
3.5 HEADINGS	3-3
3.6 INDENTED PARAGRAPHS	3-5
3.7 EMPHASIS	3-8
3.8 FOOTNOTES	3-9
3.9 DISPLAYS AND TABLES	3-9
3.10 KEEPING BLOCKS TOGETHER	3-10
3.11 NROFF COMMANDS	3-10
3.12 DATE	3-11
3.13 SIGNATURE LINE	3-11
3.14 REGISTERS	3-11
3.15 ACCENTS	3-12
3.16 USE	3-13
3.17 REFERENCES AND FURTHER STUDY	3-13
APPENDIX A	3-15

Chapter 3

USING THE -MS MACROS

3.1 INTRODUCTION

This chapter describes the **-ms** macro package of commands for producing documents on the VENIX system with the **nroff** formatting programs. As with other **roff** derived programs, text is prepared with formatting commands interspersed. However, this package, which is written in **nroff** commands, provides higher-level commands than those in the basic **nroff** program. The **-ms** commands available are listed in Appendix A. Note that they are always written in uppercase letters, while **nroff** commands are in lowercase.

3.2 PARAGRAPHS

To create a new indented paragraph when entering text, type the command **.PP** on the line preceding the sentences to be formatted. **.PP** indents the first line 5 spaces and leaves one line between paragraphs. Alternatively, the command **.LP**, which was used here, produces a left-aligned (block) paragraph with one line between paragraphs. Paragraph spacing can be changed: see below under “Registers.”

Note: you can't just begin a document with a line of text. Some **-ms** command must precede any text input. When in doubt, use **.LP** to get proper initialization, although any of the commands **.PP**, **.LP**, **.TL**, **.SH**, **.NH** are appropriate.

-MS MACROS

3.3 COVER SHEETS AND FIRST PAGES

The first command of a document can signal the general format of the first page. In particular, .RP is an overall formatting command that produces a cover sheet with title, author, author's institution, abstract and the current date. It is not necessary to have all these headings on your cover sheet, but whatever information you do use must be entered in the order specified above and preceded by the proper macro (e.g., .TL on the line before the title, and .AU before the author's name). To omit any of these sections just leave out the information and its corresponding macro.

In general **-ms** is arranged so that only one form of a document needs to be stored, containing all information. Usually the first command determines the format, and unnecessary items for that format are ignored. However, generating the cover sheet is a special process and other data and commands placed in this section may not behave as you expect. There should not be any extraneous material between the title and the end of the abstract to avoid creating format problems.

3.4 PAGE HEADINGS

By default, the **-ms** macros will print a page header with a page number (if greater than 1). A default page footer is also provided, where the date is used. The user can make minor adjustments to the page header/footer by redefining the strings LH, CH, and RH which are the left, center and right portions of the page headers, respectively; and the strings LF, CF, and RF, which are the left, center and right portions of the page footer. For more complex formats, the user can redefine the macros PT and BT, which are invoked respectively at the top and bottom of each page. The margins (taken from registers HM and FM for the top and bottom margin respectively) are normally 1 inch; the page header and footer are in the middle of that space. When redefining these macros, be careful not to change parameters such as font without resetting them to default values.

3.4.1 MULTI-COLUMN FORMATS

If you place the command ".2C" in your document, the document will be printed in double column format beginning at that point. (However, if you have a very small amount of text to be printed in double columns you may need to specify a large footer margin to insure that the text will be divided between two columns, and not just printed in one left-hand column.) The command ".1C" will go back to one-column format and also skip to a new page. The ".2C" command is actually a special case of the command

.MC [column width [gutter width]]

which makes multiple columns with the specified column and gutter width; as many columns as will fit across the page are used. Thus triple, quadruple, ... column pages can be printed. Whenever the number of columns is changed (except going from full width to some larger number of columns) a new page is started. Note that multi-column output may not be possible to put directly on your output terminal without filtering it through **col** (see **col** (1)). If the output terminal is a printer, you should always filter the output of **nroff** through the **col** utility program, and specify the type of printer by using the **-T** option.

3.5 HEADINGS

To produce a special heading, there are two commands. If you type

.NH
type section heading here
may be several lines

you will get automatically numbered section headings (1, 2, 3, ...). For example,

.NH
Care and Feeding of Department Heads

produces

1. Care and Feeding of Department Heads

-MS MACROS

Alternatively,

.SH

Care and Feeding of Directors

will print the heading with no number added:

Care and Feeding of Directors

Headings may contain more than one line of text. Every section heading, of either type, should be followed by a paragraph beginning with .PP or .LP, indicating the end of the heading.

The .NH command also supports more complex numbering schemes. If a numerical argument is given, it is taken to be a "level" number and an appropriate sub-section number is generated. Larger level numbers indicate deeper sub-sections, as in this example:

```
.NH
Erie-Lackawanna
.NH 2
Morris and Essex Division
.NH 3
Gladstone Branch
.NH 3
Montclair Branch
.NH 2
Boonton Line
```

generates:

2. Erie-Lackawanna

2.1. Morris and Essex Division

2.1.1. Gladstone Branch

2.1.2. Montclair Branch

2.2 Boonton Line

An explicit ".NH 0" will reset the level numbering to one, as here:

.NH 0
Penn Central

1. Penn Central

3.6 INDENTED PARAGRAPHS

Indented paragraphs, or paragraphs with hanging numbers, are often used for references. For example, the sequence:

.IP [1]
Text for first paragraph, typed
normally for as long as you would
like on as many lines as needed.
.IP [2]
Text for second paragraph, ...

produces

- [1] **Text for first paragraph, typed normally for as long as you would like on as many lines as needed.**
- [2] **Text for second paragraph, ...**

.IP does not automatically number the items; 1, 2, etc. need to be specified.

A series of indented paragraphs may be followed by an ordinary paragraph beginning with .PP or .LP, depending on whether you wish indenting or not. The command .LP was used here.

- MS MACROS

More sophisticated uses of .IP are also possible. If the label is omitted, for example, a plain block indent is produced.

.IP
This material will
just be turned into a
block indent suitable for quotations or
such matter.
.LP

will produce

This material will just be turned into a block indent
suitable for quotations or such matter.

.IP produces a paragraph that is indented 5 spaces from the document's left margin. If a non-standard amount of indenting is required, it may be specified after the label (in character positions) and will remain in effect until the next .PP or .LP. Thus, the general form of the .IP command contains two additional fields: the label and the indenting length. For example,

.IP first: 9
Notice the longer label, requiring larger
indenting for these paragraphs.
.IP second:
And so forth.
.LP

produces this:

first: Notice the longer label, requiring larger indenting
for these paragraphs.
second: And so forth.

It is also possible to produce multiple nested indents; the command .RS indicates that the next .IP starts from the current indentation level. Each .RE will eat up one level of indenting so you should balance .RS and .RE commands.

The .RS command should be thought of as “move right” and the .RE command as “move left.” As an example

```
.IP 1.  
Bell Laboratories  
.RS  
.IP 1.1  
Murray Hill  
.IP 1.2  
Holmdel  
.IP 1.3  
Whippany  
.RS  
.IP 1.3.1  
Madison  
.RE  
.IP 1.4  
Chester  
.RE  
.LP
```

will result in

```
1.      Bell Laboratories  
        1.1 Murray Hill  
        1.2 Holmdel  
        1.3 Whippany  
          1.3.1 Madison  
        1.4 Chester
```

All of these variations on .IP leave the right margin untouched. Sometimes, for purposes such as setting off a quotation, a paragraph indented on both right and left is required.

-MS MACROS

A single paragraph like this is obtained by preceding it with .QP. More complicated material (several paragraphs) should be bracketed with .QS and .QE.

3.7 EMPHASIS

To get "italics," which comes out as underlining on the terminal, say

```
.I  
as much text as you want  
can be typed here  
.R
```

The .R command restores the normal (usually Roman) font. If only one word is to be italicized, it may be just given on the line with the .I command,

```
.I word
```

and in this case no .R is needed to restore the previous font. **Boldface** (if available on your terminal) can be produced by

```
.B  
Text to be set in boldface  
goes here  
.R
```

As with .I, a single word can be placed in boldface by placing it on the same line as the .B command. Alternatively, the constructions '\fB', '\fI', and '\fR' can be inserted anywhere on a line to change to one of these three fonts.

```
\fBThese words will be bold\fR and \fIthese will be italicized\fR.
```

produces

These words will be bold and *these will be italicized*.

3.8 FOOTNOTES

Material placed between lines with the commands .FS (footnote start) and .FE (footnote end) — but not on the same line as the commands — will be collected, remembered and placed at the bottom of the current page*. However, there is no automatic numbering or marking of footnotes. If you want an asterisk to mark the footnote you must insert it into your paragraph text and again at the beginning of the sentence following the .FS command. By default, footnotes are 11/12th the length of normal text, but this can be changed using the .FL register (see below, in “Registers”).

3.9 DISPLAYS AND TABLES

To prepare displays of lines, such as tables, in which the lines should not be re-arranged, enclose them in the commands .DS and .DE

.DS
table lines, like the
examples here, are placed
between .DS and .DE
.DE

By default, lines between .DS and .DE are indented and left-adjusted. You can also center lines, or retain the left margin. Lines bracketed by .DS C and .DE commands are centered (and not re-arranged); lines bracketed by .DS L and .DE are left-adjusted, not indented, and not re-arranged. A plain .DS is equivalent to .DS I, which indents and left-adjusts. Thus,

these lines were
preceded by .DS C and followed by
a .DE command;

* Like this.

—MS MACROS

whereas

**these lines were preceded
by .DS L and followed by
a .DE command.**

Note that .DS C centers each line; there is a variant .DS B that makes the display into a left-adjusted block of text, and then centers that entire block. Normally a display is kept together, on one page. If you wish to have a long display which may be split across page boundaries, use .CD, .LD, or .ID in place of the commands .DS C, .DS L, or .DS I respectively. An extra argument to the .DS I or .DS command is taken as an amount to indent. Note: it is tempting to assume that .DS R will right adjust lines, but it doesn't work.

3.10 KEEPING BLOCKS TOGETHER

If you wish to keep a table or other blocks of text together on a page, there are “keep — release” commands. If a block of text, preceded by .KS and followed by .KE, does not fit on the remainder of the current page it will be placed on a new page. Lines bracketed by .DS and .DE commands are automatically kept together this way. There is also a “keep floating” command: if the block to be kept together is preceded by .KF instead of .KS and does not fit on the current page, it will be moved down through the text until the top of the next page. Thus, no large blank space will be introduced in the document.

3.11 NROFF COMMANDS

Here are some useful commands from the basic formatting program **nroff** which can be used safely with the **ms** macros:

- .bp** — begin new page.
- .br** — “break”, stop running text
from line to line.
- .sp n** — insert n blank lines.
- .na** — don't adjust right margins.

3.12 DATE

By default, documents have the current date at the bottom of each page. To force no date, say ".ND" at the beginning of your input text. To specify the date and the exact format for printing it, use the command .DA, e.g. .DA July 4, 1776. The specified date will appear at the bottom of each page. The command

.ND May 8, 1945

in the ".RP" format places the specified date on the cover sheet and nowhere else. This command line must be placed before the title.

3.13 SIGNATURE LINE

You can obtain a signature line by placing the command .SG in the document. The author's name and space for a signature will be generated by the .SG line. An argument to .SG is used as a typing identification line and placed after the signature. The .SG command is ignored in ".RP" format.

3.14 REGISTERS

Some of the registers used by -ms can be altered to change their default settings. They should be changed with .nr commands, as with

.nr VS 9p

to make the default line spacing 9 points. (Don't forget to use the scale indicator: p for points, m for ems (character widths), i for inches, etc.) Normally, these will not come into effect until the next paragraph or page. To force immediate changes in line spacing or lengths, use the corresponding nroff command, such as:

.vs 9p

for vertical spacing, or

.ll 72m

for line length.

- MS MACROS

Register	Defines	Takes effect	Default	Nroff command
VS	line spacing	next para.	12 pts	.vs
LL	line length	next para.	6"	.ll
LT	title length	next para.	6"	.lt
PD	para. spacing	next para.	0.3 VS	
PI	para. indent	next para.	5 ems	
FL	footnote length	next FS	11/12 LL	
CW	column width	next 2C	7/15 LL	
GW	intercolumn gap	next 2C	1/15 LL	
PO	page offset	next page	26/27"	.po
HM	top margin	next page	1"	
FM	bottom margin	next page	1"	

You may also alter the strings LH, CH, and RH which are the left, center and right headers respectively; and similarly LF, CF and RF which are strings in the page footer. A '%' in these strings will be replaced by the page number, which is taken on *output* from register PN to permit changing its output style. For example, saying

```
.ds CH Typing Documents on the VENIX System  
.ds CF - % -
```

prints headers and footers centered on each page.

For more complicated headers and footers the macros PT and BT can be redefined, as explained earlier.

3.15 ACCENTS

To simplify typing certain foreign words, strings representing common accent marks are defined. They precede the letter over which the mark is to appear. Here are the strings:

Input	Output	Input	Output
*' <u>me</u>	é	*a	à
*`e	è	*Ce	é
*:u	ü	*,c	ç
*^e	ê		

3.16 USE

After your document is prepared and stored on a file, you can print it on a terminal with the command*

nroff -ms filename

(many options are possible). If your document is stored in several files, just list all the filenames where we have used “filename.” If equations or tables are used, **neqn** and/or **tbl** must be invoked as preprocessors (see the *User Reference Manual* for command line examples).

3.17 REFERENCES AND FURTHER STUDY

If you have to do Greek or mathematics, see **neqn** (chapter 8 in this volume) for equation setting. To aid **neqn** users, **-ms** provides definitions of .EQ and .EN which normally center the equation and set it off slightly. An argument on .EQ is taken to be an equation number and placed in the right margin near the equation. In addition, there are three special arguments to EQ: the letters C, I, and L indicate centered (default), indented, and left adjusted equations, respectively. If there is both a format argument and an equation number, give the format argument first, as in

.EQ L (1.3a)

for a left-adjusted equation numbered (1.3a).

* If .2C was used, and your output terminal is not capable of reverse-linefeeds, have the **nroff** output piped automatically through **col**; make the first line of the input “.pi /usr/bin/col”.

-MS MACROS

Similarly, the macros .TS and .TE produce output in table format, centered, and set off from the rest of the text. A very long table with a heading may be broken across pages by beginning it with .TS H instead of .TS, and placing the command .TH in the table data after the heading. If the table has no heading repeated from page to page, just use the ordinary .TS and .TE macros. (See chapter 7.)

To learn more about **nroff** see chapter 4, NROFF TUTORIAL, and chapter 5, NROFF USER'S MANUAL, for the full details. Information on related VENIX commands is in the *User Reference Manual*. For jobs that do not seem well-adapted to **-ms**, consider other macro packages or writing your own macros. It is often far easier to write a specific macro package for tasks such as imitating particular journals than to try to adapt **-ms**.

Appendix A

List of Commands

1C Return to single column format.
2C Start double column format.
AB Begin abstract.
AE End abstract.
AI Specify author's institution.
AU Specify author.
B Begin boldface.
DA Provide the date on each page.
DE End display.
DS Start display (also CD, LD, ID).
EN End equation.
EQ Begin equation.
FE End footnote.
FS Begin footnote.

I Begin italics.

IP Begin indented paragraph.
KE Release keep.
KF Begin floating keep.
KS Start keep.

LG Increase type size.
LP Left aligned block paragraph.

ND Change or cancel date.
NH Specify numbered heading.
NL Return to normal type size.
PP Begin paragraph.

R Return to regular font (usually Roman).
RE End one level of relative indenting.
RP Use released paper format.
RS Relative indent increased one level.
SG Insert signature line.
SH Specify section heading.
SM Change to smaller type size.
TL Specify title.

UL Underline one word.

-MS MACROS

Register Names

The following register names are used by -ms internally. Independent use of these names in one's own macros may produce incorrect output. Note that no lower case letters are used in any -ms internal name.

Number registers used in -ms

:	DW	GW	HM	IQ	LL	NA	OJ	PO	T.	TV
#T	EF	H1	HT	IR	LT	NC	PD	PQ	TB	VS
.T	FC	H2	IF	IT	MF	ND	PE	PS	TC	WF
1T	FL	H3	IK	KI	MM	NF	PF	PX	TD	YE
AV	FM	H4	IM	L1	MN	NS	PI	RO	TN	YY
CW	FP	H5	IP	LE	MO	OI	PN	ST	TQ	ZN

String registers used in -ms

'	A5	CB	DW	EZ	I	KF	MR	R1	RT	TL
`	AB	CC	DY	FA	I1	KQ	ND	R2	S0	TM
^	AE	CD	E1	FE	I2	KS	NH	R3	S1	TQ
~	AI	CF	E2	FJ	I3	LB	NL	R4	S2	TS
:	AU	CH	E3	FK	I4	LD	NP	R5	SG	TT
,	B	CM	E4	FN	I5	LG	OD	RC	SH	UL
1C	BG	CS	E5	FO	ID	LP	OK	RE	SM	WB
2C	BT	CT	EE	FQ	IE	ME	PP	RF	SN	WH
A1	C	D	EL	FS	IM	MF	PT	RH	SY	WT
A2	C1	DA	EM	FV	IP	MH	PY	RP	TA	XD
A3	C2	DE	EN	FY	IZ	MN	QF	RQ	TE	XF
A4	CA	DS	EQ	HO	KE	MO	R	RS	TH	XK

CONTENTS

4.1 INTRODUCTION	4-1
4.2 LINE SPACING	4-2
4.3 FONTS	4-4
4.4 INDENTS AND LINE LENGTHS	4-6
4.5 LOCAL MOTIONS: DRAWING LINES AND CHARACTERS	4-10
4.6 STRINGS	4-13
4.7 INTRODUCTION TO MACROS	4-14
4.8 TITLES, PAGES, AND NUMBERING	4-16
4.9 NUMBER REGISTERS AND ARITHMETIC	4-20
4.10 MACROS WITH ARGUMENTS	4-22
4.11 CONDITIONALS	4-25
4.12 ENVIRONMENTS	4-27
4.13 DIVERSIONS	4-28
APPENDIX A	4-30

Chapter 4

NROFF TUTORIAL

4.1 INTRODUCTION

nroff is a text-formatting program for producing quality printed output on printers. **nroff** allows the user to simulate fonts and full control of character positions, as well as the usual features of a formatter — right-margin justification, automatic hyphenation, page titling and numbering, etc.

This chapter is an introduction to the most basic use of **nroff**. It presents enough information to enable the user to do simple formatting and to make changes to the existing packages of **nroff** commands.

For two special applications, there are programs that provide an interface to **nroff** for the majority of users. **neqn** provides an easy to learn language for formatting mathematics; the **neqn** user does not need to know any **nroff** to format mathematics. **tbl** provides the same convenience for producing tables.

For producing straight text (which may well contain mathematics or tables), there is a ‘macro package’ called **-ms** that defines formatting rules and operations for specific styles of documents, and reduces the amount of direct contact with **nroff**. (See chapter 3, USING THE **-MS** MACROS.) Typically you will find this package easier to use than **nroff**.

For the few cases where existing packages don’t meet all your needs, it is not necessary to write an entirely new set of **nroff** instructions. You can make small changes to adapt packages that already exist.

NROFF TUTORIAL

To use **nroff** you have to prepare not only the actual text you want printed, but some information that tells how you want it printed. For **nroff**, the text and the formatting information are often intertwined quite intimately. Most commands to **nroff** are placed on a line separate from the text itself, beginning with a period (one command per line). For example,

```
Some text.  
.sp 5  
Some more text.
```

will insert five spaces in between the two lines, like this:

```
Some text.
```

```
Some more text.
```

Occasionally, though, something special occurs in the middle of a line. For example, to produce

```
Area = π r2
```

you have to type

```
Area = \(*p r\^u2\^d
```

(which we will explain shortly). The backslash character `\` is used to introduce **nroff** commands and special characters within a line of text. As is often the case, the `-T` option should be used here to invoke **nroff**.

4.2 LINE SPACING

The spacing between the lines is set by default to 1/6 inch. It can be adjusted through the `.vs` command to be whatever you want (within the resolution of your output terminal, of course).

For example,

.vs .125i

sets the vertical spacing to one eighth of an inch, like this. After a few lines of text at this density, you will agree that things are rather cramped. The default line spacing is really quite adequate for most things you want to do.

You can also adjust the number of line spaces between each line with **.ls**. The command **.ls 2** sets line-spacing to twice its normal.

If you just want to leave a few blank lines somewhere in your text, use the **.sp** command. Without an argument, it will give you one extra blank line (one **.vs**, whatever that has been set to). Typically, that's more or less than you want, so **.sp** can be followed by information about how much space you want —

.sp 2i

and

.sp 2

means ‘two vertical spaces’ — two of whatever **.vs** is set to (this can also be made explicit with **.sp 2v**); **nroff** also understands decimal fractions in most places, so

.sp 1.5i

is a space of 1.5 inches. These same scale factors can be used after **.vs** to define line spacing, and in fact after most commands that deal with physical dimensions.

It should be noted that all size numbers are converted internally to ‘machine units’, which are 1/432 inch and plenty of resolution. The situation is not quite so good vertically, where resolution is 1/144 inch (but again, this is usually much better than needed).

NROFF TUTORIAL

4.3 FONTS

nroff knows about three basic fonts: the regular ("Roman") font, "italic font", and "bold" font. Text in the "italic" font is normally underlined. "Bold" font type may or may not be different than that in the regular font. If your output terminal is capable of automatically emboldening characters, and your terminal descriptor table tells **nroff** how to control it, then you can get bold face type. (Otherwise, you may wish to construct a filter to go between **nroff** and your printer, which recognizes some agreed-upon code and simulates bold font by overstriking each character.)

nroff prints in Roman unless told otherwise. To switch into bold, use the **.ft** command

.ft B

and for italics,

.ft I

To return to roman, use **.ft R**; to return to the previous font, whatever it was, use either **.ft P** or just **.ft**. The 'underline' command

.ul

causes the next input line to be underlined (producing the same output as the command for italic font). **.ul** can be followed by a number to indicate how many input lines are to be underlined.

Fonts can also be changed within a line or word with the in-line command **\f:**

boldface text

is produced by

\fBbold\fIface\fR text

If you want to do this so the previous font, whatever it was, is left undisturbed, insert extra **\fP** commands, like this:

\fBbold\fP\fIface\fP\fR text\fP

Because only the immediately previous font is remembered, you have to restore the previous font after each change or you can lose it.

Special characters have four-character names beginning with \(\), and they may be inserted anywhere. For example,

\tVENIX is a trademark . . .

is produced by

\(dgVENIX is a trademark . . .

In particular, greek letters are all of the form \(*- , where - is an upper or lower case roman letter reminiscent of the greek. Thus to get

$\Sigma(\alpha \times \beta) \rightarrow \infty$

in bare nroff we have to type

\(*S\(*a\(*mu\(*b) \(-> \if

That line is unscrambled as follows:

\(*S	Σ
((
\(*a	α
\(*mu	\times
\(*b	β
))
\(->	\rightarrow
\(if	∞

(These characters are only going to come out as well as the printer and its descriptor table can make them.)

In neqn the same effect can be achieved with the input

SIGMA (alpha times beta) -> inf

NROFF TUTORIAL

which is less concise, but clearer to the uninitiated.

Notice that each four-character name is a single character as far as **nroff** is concerned — the ‘translate’ command

.tr \(\(dg\)\(em

is perfectly clear, meaning

.tr †—

that is, to translate † into —.

4.4 INDENTS AND LINE LENGTHS

nroff starts with a line length of 6.5 inches which may not be exactly what you want. To reset the line length, use the **.ll** command, as in

.ll 6i

As with **.sp**, the actual length can be specified in several ways; **.ll 70**, for example, sets the line length to 70 ems (character-widths).

If you want to move the entire text over to the right, use the **.po** (page offset) command.

.po 1i

sets the offset one inch over.

The indent command **.in** causes the left margin to be indented by some specified amount from the page offset. (The total indentation, then, is equal to the page offset plus the given indent.) If we use **.in** to move the left margin in, and **.ll** to move the right margin to the left, we can make offset blocks of text:

```
.in 2i
.ll -2i
text to be set into a block
.ll +2i
.in -2i
```

will create a block that looks like this:

```
Pater noster qui est in caelis sanctificetur
nomen tuum; adveniat regnum tuum; fiat
voluntas tua, sicut in caelo, et in terra. ... Amen.
```

Notice the use of '+' and '-' to specify the amount of change. These change the previous setting by the specified amount, rather than just overriding it. The distinction is quite important: **.ll +2i** makes lines two inches longer; **.ll 2i** makes them two inches *long*.

With **.in**, **.ll** and **.po**, the previous value is used if no argument is specified.

To indent a single line, use the ‘temporary indent’ command **.ti**. For example, all paragraphs in this memo effectively begin with the command

```
.ti 3
```

Three of what? The default unit for **.ti**, as for most horizontally oriented commands (**.ll**, **.in**, **.po**), is ems; an em is equal to the width of one character. (In high-quality typesetting, where characters are of slightly different widths, an em is roughly equal to the width of the letter ‘m’ — hence the name.) You may find it easier to deal in character-width units than inches.

Lines can also be indented negatively if the indent is already positive:

```
.ti -3m
```

causes the next line to be moved back three character-widths. Thus to move a heading number back from a paragraph, we move the number back with a **.ti** command:

NROFF TUTORIAL

1. The committee first voted, fifteen to twelve, that
the sun shall rise at precisely 6:35 in the morning, and
set at 6:02 in the evening, except on Sundays and days beginning
with Q.

2. A movement was then made for . . .

The above paragraphs began

.in 3
.ti -3m

1.\ The committee first voted . . .

The \ after the 1. is necessary to force a single space there (otherwise it could have been padded to more).

Lines can also be centered with a .ce command.

.ce

centers the next line, and

.ce n

centers the next n lines, like this:

these lines
are centered using a .ce 2 command

Blank lines, or lines with commands on them, do not count as part of the line count given .ce.

The command .ce 0 stops the centering of lines even if a .ce command above requested it. Thus, to center a block of lines, it is easier to write

.ce 999
.
.
.ce 0

With this approach, you don't have to count the number of lines you want centered.

4.4.1 TABS

Tabs (the ASCII ‘horizontal tab’ character) can be used to produce output in columns, or to set the horizontal position of output. Typically tabs are used only in unfilled text. Tab stops are set by default every half inch from the current indent, but can be changed by the **.ta** command. To set stops every inch, for example,

```
.ta 1i 2i 3i 4i 5i 6i
```

The stops are left-justified by default (as on a typewriter); to right-justify or center the text between tabs, follow each number in the **ta** command with R or C, respectively. If you have many numbers, or if you need a more complicated table layout, don't use **nroff** directly; use the **tbl** program described in chapter 7.

For a handful of numeric columns, you can do it this way:

```
.nf
.ta 1iR 2iR 3iR
tab 1 tab 2 tab 3
tab 40 tab 50 tab 60
tab 700 tab 800 tab 900
.fi
```

to produce

1	2	3
40	50	60
700	800	900

It is also possible to fill up tabbed-over space with some character other than blanks by setting the ‘tab replacement character’ with the **.tc** command:

NROFF TUTORIAL

```
.ta 1.5i 2.5i
.tc \ru (\ru is "_")
Name tab Age tab
```

produces

Name _____ Age _____

To reset the tab replacement character to a blank, use **.tc** with no argument. (Lines can also be drawn with the **\l** command, described in the next section.)

nroff also provides a very general mechanism called ‘fields’ for setting up complicated columns. (This is used by **tbl**).

4.5 LOCAL MOTIONS: DRAWING LINES AND CHARACTERS

Remember “Area = πr^2 ”. How is the super-script done? **nroff** provides a host of commands for placing characters of any size at any place. You can use them to draw special characters or to tune your output for a particular appearance. (See also NROFF TERMINAL DESCRIPTOR TABLE FORMAT, chapter 6.) Most of these commands are straightforward, but messy to read and tough to type correctly.

If you aren’t using **neqn**, subscripts and superscripts are most easily done with the half-line local motions **\u** and **\d**. To go back up the page half a line-size, insert a **\u** at the desired place; to go down, insert a **\d**. (**\u** and **\d** should always be used in pairs, as explained below.) Thus

```
Area = \(*p r\u2\d
```

produces

```
Area =  $\pi r^2$ 
```

Sometimes the space given by **\u** and **\d** isn’t the right amount. The **\v** command can be used to request an arbitrary amount of vertical motion. The in-line command

\v'(amount)'

causes motion up or down the page by the amount specified in For example, we could create a superscript with $7\v'-.1i'3\v'.1i' = 343.$ to get

$7^3 = 343$

A minus sign causes upward motion, while no sign or a plus sign means down the page. Thus **\v'-.1i'** causes an upward vertical motion of a tenth of an inch.

There are many other ways to specify the amount of motion —

**\v'2'
\v' - 0.5m'
\v' + 2i'**

and so on are all legal. Notice that the scale specifier **i** or **m** goes inside the quotes. Any character can be used in place of the quotes; this is also true of all other **nroff** commands described in this section.

Since **nroff** does not take within-the-line vertical motions into account when figuring out where it is on the page, output lines can have unexpected positions if the left and right ends aren't at the same vertical position. Thus **\v**, like **\u** and **\d**, should always balance upward vertical motion in a line with the same amount in the downward direction.

Arbitrary horizontal motions are also available — **\h** is quite analogous to **\v**, except that the default scale factor is ems instead of line spaces. As an example,

\h' - 0.1i'

causes a backwards motion of a tenth of an inch. As a practical matter, consider printing the mathematical symbol '**>>**'. The default spacing is too wide, so **neqn** replaces this by

>\h' - 0.3m' >

to produce **>>.**

NROFF TUTORIAL

Frequently **\h** is used with the ‘width function’ **\w** to generate motions equal to the width of some character string. The construction

\w'thing'

is a number equal to the width of ‘thing’ in machine units (1/432 inch). All **nroff** computations are ultimately done in these units. To move horizontally the width of an ‘x’, we can say

\h'\w'x'u'

As we mentioned above, the default scale factor for all horizontal dimensions is **m**, ems, so here we must have the **u** for machine units, or the motion produced will be far too large. **nroff** is quite happy with the nested quotes, by the way, so long as you don’t leave any out.

As an example of this kind of construction, it is possible to bold face type through overstriking a character with a slight offset. To embolden the word “bold”, for example, you could say

bold\h' - \w'bold'u'\h'1u'.sp

That is, put out ‘bold’, move left by the width of ‘bold’, move right 1 unit, and print ‘bold’ again. (Of course there is a way to avoid typing that much input for each bold word, which we will discuss in Section 11.)

There are also several special-purpose **nroff** commands for local motion. There is **\(blank)**, which is an unpaddable character the width of a character. ‘Unpaddable’ means that it will never be widened or split across a line by line justification and filling. Another one is **\&**, which does nothing and has zero width. This is useful, for example, in entering a text line which would otherwise begin with a ‘.’.

The command **\o**, used like

\o"set of characters"

causes (up to 9) characters to be overstruck, centered on the widest. This is nice for accents, as in

`syst\o"e\(\ga"me t\o"e\(\aa'\l\o"e\(\aa"phonique`

which makes

système téléphonique

The accents are `\(ga` and `\(aa`, or `\`` and `\'`; remember that each is just one character to **nroff**.

nroff also provides a convenient facility for drawing horizontal and vertical lines of arbitrary length with arbitrary characters. `\l'1i'` draws a line one inch long, like this: . The length can be followed by the character to use if the isn't appropriate; `\l'0.5i.'` draws a half-inch line of dots: The construction `\L` is entirely analogous, except that it draws a vertical line instead of horizontal.

4.6 STRINGS

Obviously if a paper contains a large number of occurrences of a grave accent over a letter 'e', typing `\o"e\`` for each e would be a great nuisance.

Fortunately, **nroff** provides a way in which you can store an arbitrary collection of text in a 'string', and thereafter use the string name as a shorthand for its contents. Strings are one of several **nroff** mechanisms whose judicious use lets you type a document with less effort and organize it so that extensive format changes can be made with few editing changes.

A reference to a string is replaced by whatever text the string was defined as. Strings are defined with the command `.ds`. The line

`.ds e \o"e\`"`

defines the string `e` to have the value `\o"e\`"`

String names may be either one or two characters long, and are referred to by `*x` for one character names or `*(xy` for two character names. Thus to get telephone, given the definition of the string `e` as above, we can say `t*el*ephone`.

NROFF TUTORIAL

If a string must begin with blanks, define it as

```
.ds xx "      text
```

The double quote signals the beginning of the definition. There is no trailing quote; the end of the line terminates the string.

A string may actually be several lines long; if **nroff** encounters a \ at the end of any line, it is thrown away and the next line added to the current one. So you can make a long string simply by ending each line but the last with a backslash:

```
.ds xx this \
is a very \
long string
```

Strings may be defined in terms of other strings, or even in terms of themselves.

4.7 INTRODUCTION TO MACROS

Before we can go much further in **nroff**, we need to learn a bit about the macro facility. In its simplest form, a macro is just a shorthand notation quite similar to a string. Suppose we want every paragraph to start in exactly the same way — with a space and a temporary indent of two ems:

```
.sp
.ti +2m
```

Then to save typing, we would like to collapse these into one shorthand line, so that an **nroff** ‘command’ like

.PP

would be treated by **nroff** exactly as

```
.sp
.ti +2m
```

.PP is called a macro. The way we tell **nroff** what **.PP** means is to define it with the **.de** command:

```
.de PP
.sp
.ti +2m
..
```

The first line names the macro (we used ‘.PP’ for ‘paragraph’, and upper case so it wouldn’t conflict with any name that **nroff** might already know). The last line .. marks the end of the definition. In between is the text, which is simply inserted whenever **nroff** sees the ‘command’ or macro call

```
.PP
```

A macro can contain any mixture of text and formatting commands.

The definition of .PP has to precede its first use; undefined macros are simply ignored. Macro names are restricted to one or two characters.

Using macros for commonly occurring sequences of commands is critically important. Not only does it save typing, but it makes later changes much easier. Suppose we decide that the paragraph indent is too small, the vertical space is much too big, and roman font should be forced. Instead of changing the whole document, we need only change the definition of .PP to something like

```
.de PP      \" paragraph macro
.sp 2p
.ti +3m
.ft R
..
```

and the change takes effect everywhere we used .PP.

\“ is an **nroff** command that causes the rest of the line to be ignored. We use it here to add comments to the macro definition (a wise idea once definitions get complicated).

As another example of macros, consider these two which start and end a block of offset, unfilled text, like most of the examples in this paper:

NROFF TUTORIAL

```
.de BS      \> start indented block
.sp
.nf
.in +0.3i
..
.de BE      \> end indented block
.sp
.fi
.in -0.3i
..
```

Now we can surround text like

```
Copy to
John Doe
Richard Roberts
Stanley Smith
```

by the commands **.BS** and **.BE**, and it will come out as it did above. Notice that we indented by **.in +0.3i** instead of **.in 0.3i**. This way we can nest our uses of **.BS** and **.BE** to get blocks within blocks.

If later on we decide that the indent should be 0.5i, then it is only necessary to change the definitions of **.BS** and **.BE**, not the whole paper.

4.8 TITLES, PAGES, AND NUMBERING

This is an area where things get tougher, because nothing is done for you automatically. Of necessity, some of this section is a cookbook, to be copied literally until you get some experience.

Suppose you want a title at the top of each page, saying just

~~left top

center top

right top~~

You have to say what the actual title is (easy); when to print it (easy enough); and what to do at and around the title line (harder). Taking these in reverse order, first we define a macro **.NP** (for ‘new page’) to process titles and the like at the end of one page and the beginning of the next:

```
.de NP
'bp
'sp 0.5i
.tl 'left top'center top'right top'
'sp 0.3i
..
```

To make sure we're at the top of a page, we issue a 'begin page' command '**bp**', which causes a skip to top-of-page (we'll explain the '**'** shortly). Then we space down half an inch, print the title (the use of **.tl** should be self explanatory; later, we will discuss parameterizing the titles), space another 0.3 inches, and we're done.

To ask for **.NP** at the bottom of each page, we have to say something like 'when the text is within an inch of the bottom of the page, start the processing for a new page.' This is done with a 'when' command **.wh**:

```
.wh -1i NP
```

(No '.' is used before **NP**; this is simply the name of a macro, not a macro call.) The minus sign means 'measure up from the bottom of the page', so '**-1i**' means 'one inch from the bottom'.

The **.wh** command appears in the input outside the definition of **.NP**; typically the input would be

```
.de NP
...
.
.wh -1i NP
```

Now what happens? As text is actually being output, **nroff** keeps track of its vertical position on the page, and after a line is printed within one inch from the bottom, the **.NP** macro is activated. (In the jargon, the **.wh** command sets a trap at the specified place, which is 'sprung' when that point is passed.) **.NP** causes a skip to the top of the next page (that's what the '**bp**' was for), then prints the title with the appropriate margins.

NROFF TUTORIAL

Why '**bp**' and '**sp**' instead of **.bp** and **.sp**? The answer is that **.sp** and **.bp**, like several other commands, cause a break to take place. That is, all the input text collected but not yet printed is flushed out as soon as possible, and the next input line is guaranteed to start a new line of output. If we had used **.sp** or **.bp** in the **.NP** macro, this would cause a break in the middle of the current output line when a new page is started. The effect would be to print the left-over part of that line at the top of the page, followed by the next input line on a new output line. This is not what we want. Using '**'** (an apostrophe) instead of **.** for a command tells **nroff** that no break is to take place — the output line currently being filled should *not* be forced out before the space or new page.

Note that since the apostrophe character has special meaning to **nroff**, you should be careful not to begin a line of plain text with it (for example, when placing an item in single-quotes), or **nroff** will try to interpret it as a no-break command. If you begin a single-quoted item with a grave accent (which looks like an open single-quote), and close it with the apostrophe (a close single-quote), you will tend to avoid this problem.

The list of commands that cause a break is short and natural:

.bp **.br** **.ce** **.fi** **.nf** **.sp** **.in** **.ti**

All others cause no break, regardless of whether you use a **.** or a **'**. If you really need a break, add a **.br** command at the appropriate place.

One other thing to beware of — if you're changing fonts a lot, you may find that if you cross a page boundary in an unexpected font, your titles come out in that font instead of what you intended. Furthermore, the length of a title is independent of the current line length, so titles will come out at the default length of 6.5 inches unless you change it, which is done with the **.lt** command.

There are several ways to fix the problems of fonts in titles. For the simplest applications, we can change **.NP** to set the proper font for the title, then restore the previous values, like this:

```
.de NP
'bp
'sp 0.5i
.ft R      \" set title font to roman
.lt 6i      \" and length to 6 inches
.tl 'left'center'right'
.ft P      \" and to previous font
'sp 0.3i
..
..
```

This version of **.NP** does *not* work if the fields in the **.tl** command contain font changes. To cope with this situation, we need **nroff**'s ‘environment’ mechanism, which we will discuss in the section “Environments.”

To get a footer at the bottom of a page, you can modify **.NP** so it does some processing before the **'bp** command, or split the job into a footer macro invoked at the bottom margin and a header macro invoked at the top of the page. These variations are left as exercises.

Output page numbers are computed automatically as each page is produced (starting at 1), but no numbers are printed unless you ask for them explicitly. To get page numbers printed, include the character **%** in the **.tl** line at the position where you want the number to appear. For example

```
.tl - % -
```

centers the page number inside hyphens. You can set the page number at any time with either **.bp n**, which immediately starts a new page numbered **n**, or with **.pn n**, which sets the page number for the next page but doesn’t cause a skip to the new page. Again, **.bp +n** sets the page number to **n** more than its current value; **.bp** means **.bp +1**.

NROFF TUTORIAL

4.9 NUMBER REGISTERS AND ARITHMETIC

nroff has a facility for doing arithmetic, and for defining and using variables with numeric values, called number registers. Number registers, like strings and macros, can be useful in setting up a document so it is easy to change later. And of course they serve for any sort of arithmetic computation.

Like strings, number registers have one or two character names. They are set by the **.nr** command, and are referenced anywhere by **\nx** (one character name) or **\n(xy** (two character name).

There are quite a few pre-defined number registers maintained by **nroff**, among them **%** for the current page number; **nl** for the current vertical position on the page; **dy**, **mo** and **yr** for the current day, month and year; and **.s** and **.f** for the current size and font. (The font is a number from 1 to 4.) Any of these can be used in computations like any other register, but some, like **.s** and **.f**, cannot be changed with **.nr**.

As an example of the use of number registers, in the **-ms** macro package, most significant parameters are defined in terms of the values of a handful of number registers. These include the vertical spacing, and the line and title lengths. To set the vertical spacing for the following paragraphs, for example, a user may say

```
.nr VS 11
```

The paragraph macro **.PP** is defined (roughly) as follows:

```
.de PP
.vs \\n(VSp      \"
.ft R          \"
.sp 0.5v       \"
.ti +3m
..
```

This sets the font to Roman and the line spacing to whatever value is stored in the number register **VS**.

Why are there two backslashes? This is the eternal problem of how to quote a quote. When **nroff** originally reads the macro definition, it peels off one backslash to see what's coming next. To ensure that another is left in the definition when the macro is *used*, we have to put in two backslashes in the definition. If only one backslash is used, the vertical spacing will be frozen at the time the macro is defined, not when it is used.

Protection by an extra layer of backslashes is only needed for **\n**, *****, **\\$** (which we haven't come to yet), and **** itself. Things like **\s**, **\f**, **\h**, **\v**, and so on do not need an extra backslash, since they are converted by **nroff** to an internal code immediately upon being seen. The general rule is that registers used in macros must have the double backslash; otherwise, the register value interpolated is the value when the macro is first read and defined, not (as you probably want) the value when the macro is used. If you're not careful, this can be the source of many hard-to-find bugs.

Arithmetic expressions can appear anywhere that a number is expected. As a trivial example,

```
.nr VS \\n(VS-2
```

decrements VS by 2. Expressions can use the arithmetic operators **+**, **-**, *****, **/**, **%** (mod), the relational operators **>**, **>=**, **<**, **<=**, **=**, and **!=** (not equal), and parentheses.

Although the arithmetic we have done so far has been straightforward, more complicated things are somewhat tricky. First, number registers hold only integers. **nroff** arithmetic uses truncating integer division, just like Fortran. Second, in the absence of parentheses, evaluation is done left-to-right without any operator precedence (including relational operators). Thus

$$7^* - 4 + 3/13$$

becomes '**-1**'. Number registers can occur anywhere in an expression, and so can scale indicators like **p**, **i**, **m**, and so on (but no spaces). Although integer division causes truncation, each number and its scale indicator is converted to machine units (1/432 inch) before any arithmetic is done, so **1i/2u** evaluates to **0.5i** correctly.

NROFF TUTORIAL

The scale indicator **u** often has to appear when you wouldn't expect it — in particular, when arithmetic is being done in a context that implies horizontal or vertical dimensions. For example,

.ll 7/2i

would seem obvious enough — 3.5 inches. Sorry. Remember that the default units for horizontal parameters like .ll are ems. That's really '7 ems / 2 inches', and when translated into machine units, it becomes zero. How about

.ll 7i/2

Sorry, still no good — the '2' is '2 ems', so '7i/2' is small, although not zero. You *must* use

.ll 7i/2u

So again, a safe rule is to attach a scale indicator to every number, even constants.

For arithmetic done within a .nr command, there is no implication of horizontal or vertical dimension, so the default units are 'units', and 7i/2 and 7i/2u mean the same thing. Thus

```
.nr ll 7i/2  
.ll \\n(llu
```

does just what you want, so long as you don't forget the **u** on the .ll command.

4.10 MACROS WITH ARGUMENTS

The next step is to define macros that can change from one use to the next according to parameters supplied as arguments. To make this work, we need two things: first, when we define the macro, we have to indicate that some parts of it will be provided as arguments when the macro is called. Then when the macro is called we have to provide actual arguments to be plugged into the definition.

Let us illustrate by defining a macro **.SM** that will print its argument in italics (i.e. underlined). That is, the macro call

.SM NROFF

will produce *NROFF*.

The definition of **.SM** is

```
.de SM  
\fI\\$1\fP  
..
```

Within a macro definition, the symbol **\\$n** refers to the **n**th argument that the macro was called with. Thus **\\$1** is the string to be placed in the italic font when **.SM** is called.

As a slightly more complicated version, the following definition of **.SM** permits optional second and third arguments that will be printed in the regular font:

```
.de SM  
\$3\fI\\$1\fR\\$2  
..
```

Arguments not provided when the macro is called are treated as empty, so

.SM NROFF),

produces *NROFF*), while

.SM NROFF). (

produces (*NROFF*). It is convenient to reverse the order of arguments because trailing punctuation is much more common than leading.

By the way, the number of arguments with which a macro was called is available in number register **\$.**

NROFF TUTORIAL

The following macro **.BD** can be used to make bold-face text. It combines horizontal motions, width computations, and argument rearrangement.

```
.de BD
\&\\$3\f1\\$1\h' - \w'\\$1'u + 1u'\\$1\fP\\$2
..
```

The **\h** and **\w** commands need no extra backslash, as we discussed above. The **\&** is there in case the argument begins with a period.

Two backslashes are needed with the **\\\$n** commands, though, to protect one of them when the macro is being defined. Perhaps a second example will make this clearer. Consider a macro called **.SH** which produces section headings rather like those in this paper, with the section numbered automatically, and the titles in bold in a smaller size. The use is

```
.SH "Section title ..."
```

(If the argument to a macro is to contain blanks, then it must be surrounded by double quotes, unlike a string, where only one leading quote is permitted.)

Here is the definition of the **.SH** macro:

```
.nr SH 0 \" initialize section number
.de SH
.sp 0.3i
.ft B
.nr SH \\n(SH+1\" increment number
\\n(SH. \\$1 \" number. title
.sp 0.3i
.ft R
..
```

The section number is kept in number register **SH**, which is incremented each time just before it is used. (A number register may have the same name as a macro without conflict but a string may not.)

We used `\n(SH` instead of `\n(SH`. If we had used `\n(SH`, we would get the value of the register at the time the macro was defined, not at the time it was used. If that's what you want, fine, but not here.

As an example that does not involve numbers, recall our `.NP` macro which had a

```
.tl 'left'center'right'
```

We could make these into parameters by using instead

```
.tl '\\*(LT)\\*(CT)\\*(RT)
```

so the title comes from three strings called LT, CT and RT. If these are empty, then the title will be a blank line. Normally CT would be set with something like

```
.ds CT - % -
```

to give just the page number between hyphens, but a user could supply private definitions for any of the strings.

4.11 CONDITIONALS

Suppose we want the `.SH` macro to leave two extra inches of space just before section 1, but nowhere else. The cleanest way to do that is to test inside the `.SH` macro whether the section number is 1, and add some space if it is. The `.if` command provides the conditional test that we can add just before the heading line is output:

```
.if \\n(SH=1 .sp 2i \> first section only
```

The condition after the `.if` can be any arithmetic or logical expression. If the condition is logically true, or arithmetically greater than zero, the rest of the line is treated as if it were text — here a command. If the condition is false, or zero or negative, the rest of the line is skipped.

NROFF TUTORIAL

It is possible to do more than one command if a condition is true. Suppose several operations are to be done before section 1. One possibility is to define a macro .S1 and invoke it if we are about to do section 1 (as determined by an .if).

```
.de S1
--- processing for section 1 ---
.de SH
...
.if \\n(SH=1 .S1
...
..
```

An alternate way is to use the extended form of the .if, like this:

```
.if \\n(SH=1 \{--- processing
for section 1 ---\}
```

The braces \{ and \} must occur in the positions shown or you will get unexpected extra lines in your output. nroff also provides an 'if-else' construction, which we will not go into here.

A condition can be negated by preceding it with !; we get the same effect as above (but less clearly) by using

```
.if !\\n(SH>1 .S1
```

There are a handful of other conditions that can be tested with .if. For example, is the current page even or odd?

```
.if e .tl "even page title"
.if o .tl "odd page title"
```

gives facing pages different titles when used inside an appropriate new page macro.

Finally, string comparisons may be made in an .if.

```
.if  'string1'>'string2'  stuff
```

does ‘stuff’ if *string1* is the same as *string2*. The character separating the strings can be anything reasonable that is not contained in either string. The strings themselves can reference strings with *, arguments with \\$, and so on.

4.12 ENVIRONMENTS

As we mentioned, there is a potential problem when going across a page boundary: a parameter like font for a page title may well be different than that in effect in the text when the page boundary occurs. nroff provides a very general way to deal with this and similar situations. There are three ‘environments’, each of which has independently settable versions of many of the parameters associated with processing. The parameters include line spacing, font, line and title lengths, fill/nofill mode, tab stops, and even partially collected lines. Thus the titling problem may be readily solved by processing the main text in one environment and titles in a separate one with its own suitable parameters.

The command .ev *n* shifts to environment *n*; *n* must be 0, 1 or 2. The command .ev with no argument returns to the previous environment. Environment names are maintained in a stack, so calls for different environments may be nested and unwound consistently.

Suppose we say that the main text is processed in environment 0, which is where nroff begins by default. Then we can modify the new page macro .NP to process titles in environment 1 like this:

```
.de NP
.ev 1      \" shift to new environment
.lt 6i      \" set parameters here
.ft R
... any other processing ...
.ev        \" return to previous environment
..
```

NROFF TUTORIAL

It is also possible to initialize the parameters for an environment outside the .NP macro, but the version shown keeps all the processing in one place and is thus easier to understand and change.

4.13 DIVERSIONS

There are numerous occasions in page layout when it is necessary to store some text for a period of time without actually printing it. Footnotes are the most obvious example: the text of the footnote usually appears in the input well before the place on the page where it is to be printed is reached. In fact, the place where it is output normally depends on how big it is, which implies that there must be a way to process the footnote at least enough to decide its size without printing it.

nroff provides a mechanism called a diversion for doing this processing. Any part of the output may be diverted into a macro instead of being printed, and then at some convenient time the macro may be put back into the input.

The command .di **xy** begins a diversion — all subsequent output is collected into the macro **xy** until the command .di with no arguments is encountered. This terminates the diversion. The processed text is available at any time thereafter, simply by giving the command

.**xy**

The vertical size of the last finished diversion is contained in the built-in number register **dn**.

As a simple example, suppose we want to implement a ‘keep-release’ operation, so that text between the commands .KS and .KE will not be split across a page boundary (as for a figure or table). Clearly, when a .KS is encountered, we have to begin diverting the output so we can find out how big it is. Then when a .KE is read, we decide whether the diverted text will fit on the current page, and print it either there if it fits, or at the top of the next page if it doesn’t. So:

```

.de KS  \" start keep
.br   \" start fresh line
.ev 1  \" collect in new environment
.fi   \" make it filled text
.di XX \" collect in XX
..
.

.de KE \" end keep
.br   \" get last partial line
.di   \" end diversion
.if \\n(dn> =\\n(.t .bp  \" bp if doesn't fit
.nf   \" bring it back in no-fill
.XX   \" text
.ev   \" return to normal environment
..
.
```

Recall that number register **nl** is the current position on the output page. Since output was being diverted, this remains at its value when the diversion started. **dn** is the amount of text in the diversion; **.t** (another built-in register) is the distance to the next trap, which we assume is at the bottom margin of the page. If the diversion is large enough to go past the trap, the **.if** is satisfied, and a **.bp** is issued. In either case, the diverted output is then brought back with **.XX**. It is essential to bring it back in no-fill mode so **nroff** will do no further processing on it.

This is not the most general keep-release, nor is it robust in the face of all conceivable inputs, but it would require more space than we have here to write it in full generality. This section is not intended to teach everything about diversions, but to sketch out enough that you can read existing macro packages with some comprehension.

APPENDIX A:**Character Set**

The following characters have special four-character names. They will come out more or less successfully, depending on your terminal. To get the one on the left, type the four-character name on the right. For an explanation of what each character is, see table 2 of the NROFF USER'S MANUAL, chapter 5.

ff	\(ff	fi	\(fi	fl	\(fl	ffi	\(Fi
ffl	\(Fl	—	\(ru	—	\(em	¼	\(14
½	\(12	¾	\(34	¢	\(ct	-	\(hy
©	\(co	°	\(de	†	\(dg	'	\(fm
®	\(rg	●	\(bu	□	\(sq	*	\(**
+	\(pl	—	\(mi	×	\(mu	÷	\(di
=	\(eq	≡	\(==	≥	\(>=	≤	\(<=
≠	\(!=	±	\(+-	¬	\(no	/	\(sl
~	\(ap	≈	\(=	∞	\(pt	▽	\(gr
→	\(->	←	\(<-	↑	\(ua	↓	\(da
∫	\(is	∂	\(pd	∞	\(if	√	\(sr
⊂	\(sb	⊃	\(sp	∪	\(cu	∩	\(ca
⊆	\(ib	⊇	\(ip	€	\(mo	›	\(es
`	\(aa	`	\(ga	○	\(ci	Ⓐ	\(bs
§	\(sc	‡	\(dd	▀	\(lh	▀	\(rh
{	\(lt)	\(rt	└	\(lc]	\(rc
\(lb)	\(rb)	└	\(lf]	\(rf
{	\(lk	}	\(rk	└	\(bv	underline	\(ts
	\(br		\(or	—	\(ul		\(rn

NROFF TUTORIAL

These four characters also have two-character names. The ' is the apostrophe on terminals; the ` is the other quote mark.

' \` \^ - - - _

For greek, precede the roman letter by \(* to get the corresponding greek; for example, \(*a is α .

a	b	g	d	e	z	y	h	i	k	l	m	n	c	o	p	r	s	t	u	f	x	q	w
α	β	γ	δ	ϵ	ζ	η	θ	ι	κ	λ	μ	ν	ξ	σ	π	ρ	σ	τ	v	ϕ	χ	ψ	ω

A	B	G	D	E	Z	Y	H	I	K	L	M	N	C	O	P	R	S	T	U	F	X	Q	W
A	B	Γ	Δ	E	Z	Y	Θ	I	K	Λ	M	N	Ξ	O	Π	P	Σ	T	T	Φ	X	Ψ	Ω

CONTENTS

5.1 INTRODUCTION	5-1
5.2 USAGE	5-1
5.3 SUMMARY AND INDEX	5-3
5.4 GENERAL EXPLANATION	5-14
5.5 FONTS	5-17
5.6 PAGE CONTROL	5-17
5.7 TEXT FILLING, ADJUSTING, AND CENTERING	5-19
5.8 VERTICAL SPACING	5-21
5.9 LINE LENGTH AND INDENTING	5-23
5.10 MACROS, STRINGS, DIVERSION, AND POSITION TRAPS	5-24
5.11 NUMBER REGISTERS	5-29
5.12 TABS, LEADERS, AND FIELDS	5-31
5.13 INPUT/OUTPUT CONVENTIONS AND CHARACTER TRANSLATIONS	5-33
5.14 LOCAL MOTIONS AND THE WIDTH FUNCTION	5-36
5.15 OVERSTRIKE, BRACKET, LINE-DRAWING, AND ZERO-WIDTH FUNCTIONS	5-37
5.16 HYPHENATION	5-39
5.17 THREE PART TITLES	5-40
5.18 OUTPUT LINE NUMBERING	5-41
5.19 CONDITIONAL ACCEPTANCE OF INPUT	5-42
5.20 ENVIRONMENT SWITCHING	5-44

5.21	INSERTIONS FROM THE STANDARD INPUT	5-45
5.22	INPUT/OUTPUT FILE SWITCHING	5-46
5.23	MISCELLANEOUS	5-46
5.24	OUTPUT AND ERROR MESSAGES	5-47
	TABLE I	5-49

Chapter 5

NROFF USER'S MANUAL

5.1 INTRODUCTION

Nroff is a text processor that formats text for typewriter-like terminals. It accepts lines of text interspersed with lines of format control information and formats the text into a printable, paginated document having a user-designed style. Nroff offers unusual freedom in document styling, including: arbitrary style headers and footers; arbitrary style footnotes; multiple automatic sequence numbering for paragraphs, sections, etc; multiple column output; dynamic font and point-size control; arbitrary horizontal and vertical local motions at any point; and a family of automatic overstriking, bracket construction, and line drawing functions.

Conditional input is provided that enables the user to embed input expressly destined for either program. Nroff can prepare output directly for a variety of terminal types and is capable of utilizing the full resolution of each terminal.

5.2 USAGE

The general form of invoking Nroff at the command level is

nroff *options files*

where *options* represents any of a number of option arguments and *files* represents the list of files containing the document to be formatted. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input. If no file names are given input is taken from the standard input. The options, which may appear in any order so long as they appear before the files, are:

NROFF USER'S MANUAL

Option	Effect
-o <i>list</i>	Print only pages whose page numbers appear in <i>list</i> , which consists of comma-separated numbers and number ranges. A number range has the form <i>N-M</i> and means pages <i>N</i> through <i>M</i> ; a initial <i>-N</i> means from the beginning to page <i>N</i> ; and a final <i>N-</i> means from <i>N</i> to the end.
-n <i>N</i>	Number first generated page <i>N</i> .
-s <i>N</i>	Stop every <i>N</i> pages. Nroff will halt prior to every <i>N</i> pages (default <i>N</i> =1) to allow paper loading or changing, and will resume upon receipt of a newline.
-m <i>name</i>	Prepends the macro file <code>/usr/lib/tmac.<i>name</i></code> to the input <i>files</i> .
-r <i>a</i> <i>N</i>	Register <i>a</i> (one-character) is set to <i>N</i> .
-h	Output tabs used during horizontal spacing to speed output as well as reduce output byte count. Device tab setting assumed to be every 8 nominal character widths. The default settings of input (logical) tabs is also initialized to every 8 nominal character widths.
-i	Read standard input after the input files are exhausted.
-q	Invoke the simultaneous input-output mode of the rd request.
-T <i>name</i>	Specifies the name of the output terminal type. Currently defined names are 37 for the (default) Model 37 Teletype®, tn300 for the GE TermiNet 300 (or any terminal without half-line capabilities), 300S for the DASI-300S, 300 for the DASI-300, and 450 for the DASI-450 (Diablo Hyterm).
-e	Produce equally-spaced words in adjusted lines, using full terminal resolution.
-z	Efficiently suppresses formatted output. Only message output will occur.

Each option is invoked as a separate argument; for example,

```
nroff -o4,8-10 -T300S -mabc file1 file2
```

requests formatting of pages 4, 8, 9, and 10 of a document contained in the files named *file1* and *file2*, specifies the output terminal as a DASI-300S, and invokes the macro package *abc*.

Various pre- and post-processors are available for use with Nroff. These include the equation preprocessors **neqn** and the table-construction preprocessor **tbl**. A reverse-line postprocessor **col** is available for multiple-column Nroff output on terminals without reverse-line ability; **col** expects the Model 37 Teletype escape sequences that Nroff produces by default.

The remainder of this chapter consists of: a Summary and Index, and a Reference section.

5.3 SUMMARY AND INDEX

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES#	EXPLANATION
---------------------	----------------------	-----------------------	---------------	--------------------

5.3.1 Font and Character Size Control

.ss <i>N</i>	12/36 em	ignored	E	Space-character size set to <i>N</i> /36em. [†]
.cs <i>FNM</i>	off	-	P	Constant character space (width) mode (font <i>F</i>). [†]
.bd <i>F N</i>	off	-	P	Embolden font <i>F</i> by <i>N</i> –1 units. [†]
.bd S <i>F N</i>	off	-	P	Embolden Special Font when current font is <i>F</i> . [†]
.ft <i>F</i>	Roman	previous	E	Change to font <i>F</i> = <i>x</i> , <i>xx</i> , or 1–4. Also \fx, \f(xx, \f <i>N</i> .

Notes are explained at the end of 5.3.17

NROFF USER'S MANUAL

.fp $N F$ R,I,B,S ignored - Font named F mounted on physical position $1 \leq N \leq 4$.

5.3.2 Page Control

.pl $\pm N$	11 in	11 in	v	Page length.
.bp $\pm N$	$N=1$	-	B‡,v	Eject current page; next page number N .
.pn $\pm N$	$N=1$	ignored	-	Next page number N .
.po $\pm N$	0;26/27 in	previous	v	Page offset.
.ne N	-	$N=1V$	D,v	Need N vertical space (V = vertical spacing).
.mk R	none	internal	D	Mark current vertical place in register R .
.rt $\pm N$	none	internal	D,v	Return (<i>upward only</i>) to marked vertical place.

5.3.3 Text Filling, Adjusting, and Centering

.br	-	-	B	Break.
.fi	fill	-	B,E	Fill output lines.
.nf	fill	-	B,E	No filling or adjusting of output lines.

† No effect in Nroff.

‡ The use of “ ‘ ’ ” as control character (instead of “.”) suppresses the break function.

.ad <i>c</i>	adj,both	adjust	E	Adjust output lines with mode <i>c</i> .
.na	adjust	-	E	No output line adjusting.
.ce <i>N</i>	off	<i>N</i> =1	B,E	Center following <i>N</i> input text lines.

5.3.4 Vertical Spacing

.vs <i>N</i>	1/6in;12pts	previous	E,p	Vertical base line spacing (<i>V</i>).
.ls <i>N</i>	<i>N</i> =1	previous	E	Output <i>N</i> -1 <i>V</i> s after each text output line.
.sp <i>N</i>	-	<i>N</i> =1 <i>V</i>	B,v	Space vertical distance <i>N</i> in either direction.
.sv <i>N</i>	-	<i>N</i> =1 <i>V</i>	v	Save vertical distance <i>N</i> .
.os	-	-	-	Output saved vertical distance.
.ns	space	-	D	Turn no-space mode on.
.rs	-	-	D	Restore spacing; turn no-space mode off.

5.3.5 Line Length and Indenting

.ll $\pm N$	6.5 in	previous	E,m	Line length.
.in $\pm N$	<i>N</i> =0	previous	B,E,m	Indent.
.ti $\pm N$	-	ignored	B,E,m	Temporary indent.

NROFF USER'S MANUAL

5.3.6 Macros, Strings, Diversion, and Position Traps

.de <i>xx yy</i>	-	.yy = ..	-	Define or redefine macro <i>xx</i> ; end at call of <i>yy</i> .
.am <i>xx yy</i>	-	.yy = ..	-	Append to a macro.
.ds <i>xx string</i>	-	ignored	-	Define a string <i>xx</i> containing <i>string</i> .
.as <i>xx string</i>	-	ignored	-	Append <i>string</i> to string <i>xx</i> .
.rm <i>xx</i>	-	ignored	-	Remove request, macro, or string.
.rn <i>xx yy</i>	-	ignored	-	Rename request, macro, or string <i>xx</i> to <i>yy</i> .
.di <i>xx</i>	-	end	D	Divert output to macro <i>xx</i> .
.da <i>xx</i>	-	end	D	Divert and append to <i>xx</i> .
.wh <i>N xx</i>	-	-	v	Set location trap; negative is w.r.t. page bottom.
.ch <i>xx N</i>	-	-	v	Change trap location.
.dt <i>N xx</i>	-	off	D,v	Set a diversion trap.
.it <i>N xx</i>	-	off	E	Set an input-line count trap.
.em <i>xx</i>	none	none	-	End macro is <i>xx</i> .

5.3.7 Number Registers

.nr <i>R</i> ± <i>NM</i>	-	u	Define and set number register <i>R</i> ; auto-increment by <i>M</i> .
--------------------------	---	---	--

.af *R c* arabic - - Assign format to register *R* (*c*=**1**, **i**, **I**, **a**, **A**).

.rr *R* - - - Remove register *R*.

5.3.8 Tabs, Leaders, and Fields

.ta *Nt ...* 0.8; 0.5in none E,m Tab settings; *left* type, unless *t*=**R**(right), **C**(centered).

.tc *c* none none E Tab repetition character.

.lc *c* . none E Leader repetition character.

.fc *a b* off off - Set field delimiter *a* and pad character *b*.

5.3.9 Input and Output Conventions and Character Translations

.ec *c* \ \ - Set escape character.

.eo on - - Turn off escape character mechanism.

.lg *N* -;on on - Ligature mode on if *N*>0.

.ul *N* off *N*=1 E Underline *N* input lines.

.cu *N* off *N*=1 E Continuous underline.

.uf *F* Italic Italic - Underline font set to *F* (to be switched to by **ul**).

.cc *c* . . E Set control character to *c*.

.c2 *c* ' ' E Set nobreak control character to *c*.

.tr *abcd....* none - O Translate *a* to *b*, etc. on output.

NROFF USER'S MANUAL

5.3.10 Hyphenation

.nh	hyphenate	-	E	No hyphenation.
.hy <i>N</i>	hyphenate	hyphenate	E	Hyphenate; <i>N</i> = mode.
.hc <i>c</i>	\%	\%	E	Hyphenation indicator character <i>c</i> .
.hw <i>word</i> ...	ignored	-	-	Exception words.

5.3.11 Three Part Titles

.tl 'left'center'right'	-	-	-	Three part title.
.pc <i>c</i>	%	off	-	Page number character.
.lt $\pm N$	6.5 in	previous	E,m	Length of title.

5.3.12 Output Line Numbering

.nm $\pm NMSI$	-	off	E	Number mode on or off, set parameters.
.nn <i>N</i>	-	<i>N</i> =1	E	Do not number next <i>N</i> lines.

5.3.13 Conditional Acceptance of Input

.if <i>c anything</i>	-	-	-	If condition <i>c</i> true, accept <i>anything</i> as input, for multi-line use \{ <i>anything</i> \}.
.if ! <i>c anything</i>	-	-	-	If condition <i>c</i> false, accept <i>anything</i> .
.if <i>N anything</i>	-	u	-	If expression <i>N</i> > 0, accept <i>anything</i> .

.if <i>N anything</i>	-	u	If expression <i>N</i> ≤ 0 , accept <i>anything</i> .
.if ' <i>string1</i> ' ' <i>string2</i> ' <i>anything</i>	-		If <i>string1</i> identical to <i>string2</i> , accept <i>anything</i> .
.if '!' <i>string1</i> ' ' <i>string2</i> ' <i>anything</i>	-		If <i>string1</i> not identical to <i>string2</i> , accept <i>anything</i> .
.ie <i>c anything</i>	-	u	If portion of if-else; all above forms (like if).
.el <i>anything</i>	-	-	Else portion of if-else.

5.3.14 Environment Switching

.ev <i>N</i>	<i>N=0</i>	previous	-	Environment switched (<i>push down</i>).
---------------------	------------	----------	---	--

5.3.15 Insertions from the Standard Input

.rd <i>prompt</i>	-	<i>prompt=BEL</i>	-	Read insertion.
.ex	-	-	-	Exit from Nroff.

5.3.16 Input/Output File Switching

.so <i>filename</i>	-	-	-	Switch source file (<i>push down</i>).
.nx <i>filename</i>		<i>end-of-file</i>	-	Next file.
.pi <i>program</i>	-	-	-	Pipe output to <i>program</i> .

5.3.17 Miscellaneous

.mc <i>c N</i>	-	<i>off</i>	E,m	Set margin character <i>c</i> and separation <i>N</i> .
-----------------------	---	------------	------------	---

NROFF USER'S MANUAL

.tm <i>string</i>	-	newline	-	Print <i>string</i> on terminal (UNIX standard message output).
.ig <i>yy</i>	-	. <i>yy</i> = ..	-	Ignore till call of <i>yy</i> .
.pm <i>t</i>	-	all	-	Print macro names and sizes; if <i>t</i> present, print only total of sizes.
.fl	-	-	B	Flush output buffer.

Notes -

- B Request normally causes a break.
D Mode or relevant parameters associated with current diversion level.
E Relevant parameters are a part of the current environment.
O Must stay in effect until logical output.
P Mode must be still or again in effect at the time of physical output.
v,p,m,u Default scale indicator; if not specified, scale indicators are *ignored*.

Alphabetical Request and Section Number Cross Reference

ad	3	de	6	ft	1	mc	17	pl	2	sv	4
af	7	di	6	hc	10	mk	2	pm	17	ta	8
am	6	ds	6	hw	10	na	3	pn	3	tc	8
as	6	dt	6	hy	10	ne	2	po	3	ti	5
bd	1	ec	9	ie	13	nf	3	ps	2	tl	11
bp	2	el	13	if	13	nh	10	rd	15	tm	17
br	3	em	6	ig	17	nm	12	rm	6	tr	9
c2	9	eo	9	in	5	nn	12	rn	6	uf	9
cc	9	ev	14	it	6	nr	7	rr	7	l	9
ce	3	ex	15	lc	8	ns	4	urs	4	vs	4
ch	6	fc	8	lg	9	nx	16	rt	2	wh	6
cs	1	fi	3	ll	5	os	4	so	16		
cu	9	fl	17	ls	4	pc	11	sp	4		
da	6	fp	1	lt	11	pi	16	ss	1		

5.3.18 Escape Sequences for Characters, Indicators, and Functions

<i>Sequence</i>	<i>Meaning</i>
\\\	\ (to prevent or delay the interpretation of \)
\e	Printable version of the <i>current</i> escape character.
\`	' (acute accent); equivalent to \(\aa
\`	` (grave accent); equivalent to \(\ga
\-	- Minus sign in the <i>current</i> font
\.	Period (dot) (see d_e)
\(space)	Unpaddable space-size space character
\0	Digit width space
\`	1/6 em narrow space character (zero width in Nroff)
\^	1/12 em half-narrow space character (zero width in Nroff)
\&	Non-printing, zero width character
\!	Transparent line indicator
\"	Beginning of comment
\\$N	Interpolate argument $1 \leq N \leq 9$
\%	Default optional hyphenation character
\(xx	Character named <i>xx</i>
*(<i>x</i> , *(<i>xx</i>	Interpolate string <i>x</i> or <i>xx</i>
\a	Non-interpreted leader character
\b'abc...'	Bracket building function
\c	Interrupt text processing
\d	Forward (down) 1/2 em vertical motion (1/2 line in Nroff)
\fx,\f(<i>xx</i> ,\f <i>N</i>	Change to font named <i>x</i> or <i>xx</i> , or position <i>N</i>
\h'N'	Local horizontal motion; move right <i>N</i> (<i>negative left</i>)
\k <i>x</i>	Mark horizontal <i>input</i> place in register <i>x</i>
\l'Nc'	Horizontal line drawing function (optionally with <i>c</i>)
\L'Nc'	Vertical line drawing function (optionally with <i>c</i>)
\nx,\n(<i>xx</i>	Interpolate number register <i>x</i> or <i>xx</i>
\o'abc...'	Overstrike characters <i>a</i> , <i>b</i> , <i>c</i> , ...
\p	Break and spread output line
\r	Reverse 1 em vertical motion (reverse line in Nroff)
\s <i>N</i> ,\s\pm <i>N</i>	Point-size change function
\t	Non-interpreted horizontal tab
\u	Reverse (up) 1/2 em vertical motion (1/2 line in Nroff)

NROFF USER'S MANUAL

\v'N'	Local vertical motion; move down <i>N</i> (<i>negative up</i>)
\w' <i>string</i> '	Interpolate width of <i>string</i>
\x'N'	Extra line-space function (<i>negative before, positive after</i>)
\zc	Print <i>c</i> with zero width (without spacing)
\{	Begin conditional input
\}	End conditional input
\(newline)	Concealed (ignored) newline
\ <i>X</i>	<i>X</i> , any character <i>not</i> listed above

The escape sequences \\, \., \", \\$, *, @, \n, \t, and \newline are interpreted in *copy mode*.

5.3.19 Predefined General Number Registers

Register	
Name	Description
%	Current page number.
ct	Character type (set by <i>width</i> function).
dl	Width (maximum) of last completed diversion.
dn	Height (vertical size) of last completed diversion.
dw	Current day of the week (1-7).
dy	Current day of the month (1-31).
hp	Current horizontal place on <i>input</i> line.
ln	Output line number.
mo	Current month (1-12).
nl	Vertical position of last printed text base-line.
sb	Depth of string below base line (generated by <i>width</i> function).
st	Height of string above base line (generated by <i>width</i> function).
yr	Last two digits of current year.

5.3.20 Predefined Read-Only Number Registers

Register	
Name	Description
.\$	Number of arguments available at the current macro level.
.A	Set to 1.
.H	Available horizontal resolution in basic units.
.P	Contains the current line spacing parameter.
.P	Set to 1 if the current page is being printed, and zero otherwise.
.T	Set to 1 if -T option used;
.V	Available vertical resolution in basic units.
.a	Post-line extra line-space most recently utilized using \x'N'.
.c	Number of <i>lines</i> read from current input file.
.d	Current vertical place in current diversion; equal to nl, if no diversion.
.f	Current font as physical quadrant (1-4).
.j	A number representing the current adjustment mode and type.

NROFF USER'S MANUAL

- .k Contains the horizontal size of the text portion of the current partially collected output line.
- .h Text base-line high-water mark on current page or diversion.
- .i Current indent.
- .l Current line length.
- .n Length of text portion on previous output line.
- .o Current page offset.
- .p Current page length.
- .s Current point size.
- .t Distance to the next trap.
- .u Equal to 1 in fill mode and 0 innofill mode.
- .v Current vertical line spacing.
- .w Width of previous character.
- .x Reserved version-dependent register.
- .y Reserved version-dependent register.
- .z Name of current diversion.

5.4 GENERAL EXPLANATION

5.4.1 Form of Input

Input consists of *text lines*, which are destined to be printed, interspersed with *control lines*, which set parameters or otherwise control subsequent processing. Control lines begin with a *control character* — normally . (period) or ' (acute accent) — followed by a one or two character name that specifies a basic *request* or the substitution of a user-defined *macro* in place of the control line. The control character ' suppresses the *break* function — the forced output of a partially filled line — caused by certain requests. The control character may be separated from the request/macro name by white space (spaces and/or tabs) for esthetic reasons. Names must be followed by either space or newline. Control lines with unrecognized names are ignored.

Various special functions may be introduced anywhere in the input by means of an *escape* character, normally \. For example, the function \nR causes the interpolation of the contents of the *number register R* in place of the function; here R is either a single character name as in \nx, or left-parenthesis-introduced, two-character name as in \n(xx.

5.4.2 Formatter and Device Resolution

Nroff internally uses 240 units/inch, corresponding to the least common multiple of the horizontal and vertical resolutions of various typewriter-like output devices. Nroff rounds numerical input to the actual resolution of the output device indicated by the **-T** option (default Model 37 Teletype).

5.4.3 Numerical Parameter Input

Nroff accepts numerical input with the appended scale indicators shown in the following table, where S is the current type size in points, V is the current vertical line spacing in basic units, and C is a *nominal character width* in basic units.

Scale Indicator	Number of basic units	
i	Inch	
c	Centimeter	$240 \times 50/127$
P	Pica = 1/6 inch	$240/6$
m	Em = S points	C
n	En = Em/2	
p	Point = 1/72 inch	$240/72$
u	Basic unit	1
v	Vertical line space	V
none	Default, see below	

In Nroff, *both* the em and the en are taken to be equal to the C , which is output-device dependent; common values are 1/10 and 1/12 inch. Actual character widths in Nroff need not be all the same and constructed characters such as **->** (**-**) are often extra wide. The default scaling is ems for the horizontally-oriented requests and functions **ll**, **in**, **ti**, **ta**, **lt**, **po**, **mc**, **\h**, and **\l**; V s for the vertically-oriented requests and functions **pl**, **wh**, **ch**, **dt**, **sp**, **sv**, **ne**, **rt**, **\v**, **\x**, and **\L**; **p** for the **vs** request; and **u** for the requests **mr**, **if**, and **ie**. All other requests ignore any scale indicators. When a number register containing an already appropriately scaled number is interpolated to provide numerical input, the unit scale indicator **u** may need to be appended to prevent an additional inappropriate default scaling. The number, N , may be specified in decimal-fraction form but the parameter finally stored is rounded to an integer number of basic units.

NROFF USER'S MANUAL

The *absolute position* indicator \sim may be prepended to a number N to generate the distance to the vertical or horizontal place N . For vertically-oriented requests and functions, $\sim N$ becomes the distance in basic units from the current vertical place on the page or in a *diversion* to the the vertical place N . For *all* other requests and functions, $\sim N$ becomes the distance from the current horizontal place on the *input* line to the horizontal place N . For example,

```
.sp ~ 3.2c
```

will space *in the required direction* to 3.2 centimeters from the top of the page.

5.4.4 Numerical Expressions

Wherever numerical input is expected an expression involving parentheses, the arithmetic operators $+$, $-$, $/$, $*$, $\%$ (mod), and the logical operators $<$, $>$, $<=$, $>=$, $=$ (or $==$), $\&$ (and), $:$ (or) may be used. Except where controlled by parentheses, evaluation of expressions is left-to-right; there is no operator precedence. In the case of certain requests, an initial $+$ or $-$ is stripped and interpreted as an increment or decrement indicator respectively. In the presence of default scaling, the desired scale indicator must be attached to *every* number in an expression for which the desired and default scaling differ. For example, if the number register **x** contains 2 and the current point size is 10, then

```
.ll (4.25i+\nxP+3)/2u
```

will set the line length to 1/2 the sum of 4.25 inches + 2 picas + 30 points.

5.4.5 Notation

Numerical parameters are indicated in this manual in two ways. $\pm N$ means that the argument may take the forms N , $+N$, or $-N$ and that the corresponding effect is to set the affected parameter to N , to increment it by N , or to decrement it by N respectively. Plain N means that an initial algebraic sign is *not* an increment indicator, but merely the sign of N . Generally, unreasonable numerical input is either ignored or truncated to a reasonable value. For example, most requests expect to set parameters to non-negative values; exceptions are **sp**, **wh**, **ch**, **nr**, and **if**. The requests **ft**, **po**, **vs**, **ls**, **ll**, **in**, and **lt** restore the *previous* parameter value in the *absence* of an argument.

Single character arguments are indicated by single lower case letters and one/two character arguments are indicated by a pair of lower case letters. Character string arguments are indicated by multi-character mnemonics.

5.5 FONTS

The default mounted fonts are Roman (**R**), Italic (**I**), (underline) and Bold (**B**).

5.6 PAGE CONTROL

Top and bottom margins are *not* automatically provided; it is conventional to define two *macros* and to set *traps* for them at vertical positions 0 (top) and $-N$ (N from the bottom). A pseudo-page transition onto the *first* page occurs either when the first *break* occurs or when the first *non-diverted* text processing occurs. Arrangements for a trap to occur at the top of the first page must be completed before this transition. In the following, references to the *current diversion* mean that the mechanism being described works during both ordinary and diverted output (the former considered as the top diversion level).

The physical limitations on Nroff output are output-device dependent.

REQUEST	INITIAL	IF NO			EXPLANATION
FORM	VALUE	ARGUMENT	NOTES		
.pl $\pm N$	11 in	11 in	v		Page length set to $\pm N$. The internal limitation is about 136 inches. The current page length is available in the .p register.
.bp $\pm N$	$N=1$	-	B*,v		Begin page. The current page is ejected and a new page is begun. If $\pm N$ is given, the new page number will be $\pm N$. Also see request ns.

* The use of “ ‘ ’ ” as control character (instead of “.”) suppresses the break function.

NROFF USER'S MANUAL

.pn $\pm N$	$N=1$	ignored	-	Page number. The next page (when it occurs) will have the page number $\pm N$. A pn must occur before the initial pseudo-page transition to effect the page number of the first page. The current page number is in the % register.
.po $\pm N$	0	previous	v	Page offset. The current <i>left margin</i> is set to $\pm N$. The maximum (line-length) + (page-offset) is about 7.54 inches. The current page offset is available in the .o register.
.ne N	-	$N=1V$	D,v	Need N vertical space. If the distance, D , to the next trap position is less than N , a forward vertical space of size D occurs, which will spring the trap. If there are no remaining traps on the page, D is the distance to the bottom of the page. If $D < V$, another line could still be output and spring the trap. In a diversion, D is the distance to the <i>diversion trap</i> , if any, or is very large.
.mk R	none	internal	D	Mark the <i>current</i> vertical place in an internal register (both associated with the current diversion level), or in register R , if given. See rt request.
.rt $\pm N$	none	internal	D,v	Return <i>upward only</i> to a marked vertical place in the current diversion. If $\pm N$ (w.r.t. current place) is given, the place is $\pm N$ from the top of the page or diversion or, if N is absent, to a place marked by a previous mk .

Note that the **sp** request may be used in all cases instead of **rt** by spacing to the absolute place stored in an explicit register; e. g. using the sequence .mk *R*sp
| \nRu.

5.7 TEXT FILLING, ADJUSTING, AND CENTERING

5.7.1 Filling and Adjusting

Normally, words are collected from input text lines and assembled into an output text line until some word doesn't fit. An attempt is then made hyphenate the word in effort to assemble a part of it into the output line. The spaces between the words on the output line are then increased to spread out the line to the current *line length* minus any current *indent*. A *word* is any string of characters delimited by the *space* character or the beginning/end of the input line. Any adjacent pair of words that must be kept together (neither split across output lines nor spread apart in the adjustment process) can be tied together by separating them with the *unpaddable space* character “\ ” (backslash-space). In Nroff, they are normally nonuniform because of quantization to character-size spaces; however, the command line option **-e** causes uniform spacing with full output device resolution. Filling, adjustment, and hyphenation can all be prevented or controlled. The *text length* on the last line output is available in the **.n** register, and text base-line position on the page for this line is in the **nl** register. The text base-line high-water mark (lowest place) on the current page is in the **.h** register.

An input text line ending with ., ?, or ! is taken to be the end of a *sentence*, and an additional space character is automatically provided during filling. Multiple inter-word space characters found in the input are retained, except for trailing spaces; initial spaces also cause a *break*.

When filling is in effect, a \p may be imbedded or attached to a word to cause a *break* at the *end* of the word and have the resulting output line *spread out* to fill the current line length.

NROFF USER'S MANUAL

A text input line that happens to begin with a control character can be made to not look like a control line by prefacing it with the non-printing, zero-width filler character \&. Still another way is to specify output translation of some convenient character into the control character using *tr*.

5.7.2 Interrupted Text

The copying of a input line in *nofill* (non-fill) mode can be *interrupted* by terminating the partial line with a \c. The *next* encountered input text line will be considered to be a continuation of the same line of input text. Similarly, a word within *filled* text may be interrupted by terminating the word (and line) with \c; the next encountered text will be taken as a continuation of the interrupted word. If the intervening control lines cause a break, any partial line will be forced out along with any partial word.

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES	EXPLANATION
.br	-	-	B	Break. The filling of the line currently being collected is stopped and the line is output without adjustment. Text lines beginning with space characters and empty text lines (blank lines) also cause a break.
.fi	fill on	-	B,E	Fill subsequent output lines. The register .u is 1 in fill mode and 0 in nofill mode.
.nf	fill on	-	B,E	Nofill. Subsequent output lines are <i>neither</i> filled <i>nor</i> adjusted. Input text lines are copied directly to output lines <i>without regard</i> for the current line length.
.ad c	adj,both	adjust	E	Line adjustment is begun. If fill mode is not on, adjustment will be deferred until fill mode is back on. If the type indicator <i>c</i> is present, the adjustment type is changed as shown in the following table.

Indicator	Adjust Type
l	adjust left margin only
r	adjust right margin only
c	center
b or n	adjust both margins
absent	unchanged

.na	adjust	-	E	Noadjust. Adjustment is turned off; the right margin will be ragged. The adjustment type for ad is not changed. Output line filling still occurs if fill mode is on.
.ce <i>N</i>	off	<i>N</i> =1	B,E	Center the next <i>N</i> input text lines within the current (line-length minus indent). If <i>N</i> =0, any residual count is cleared. A break occurs after each of the <i>N</i> input lines. If the input line is too long, it will be left adjusted.

5.8 VERTICAL SPACING

5.8.1 Base-line Spacing

The vertical spacing (*V*) between the base-lines of successive output lines can be set using the vs request. The current *V* is available in the .v register. Multiple-*V* line separation (e.g. double spacing) may be requested with ls.

5.8.2 Extra Line-space

If a word contains a vertically tall construct requiring the output line containing it to have extra vertical space before and/or after it, the *extra-line-space* function \x'N' can be imbedded in or attached to that word. In this and other functions having a pair of delimiters around their parameter (here '), the delimiter choice is arbitrary, except that it can't look like the continuation of a number expression for *N*. If *N* is negative, the output line containing the word will be preceded by *N* extra vertical space; if *N* is positive, the output line

NROFF USER'S MANUAL

containing the word will be followed by N extra vertical space. If successive requests for extra space apply to the same line, the maximum values are used. The most recently utilized post-line extra line-space is available in the .a register.

5.8.3 Blocks of Vertical Space

A block of vertical space is ordinarily requested using **sp**, which honors the *no-space* mode and which does not space *past* a trap. A contiguous block of vertical space may be reserved using **sv**.

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES	EXPLANATION
.vs N	1/6in	previous	E, p	Set vertical baseline spacing size V . Transient <i>extra</i> vertical space available with \x'N' (see above).
.ls N	$N = 1$	previous	E	<i>Line</i> spacing set to $\pm N$. $N - 1$ Vs (<i>blank lines</i>) are appended to each output text line. Appended blank lines are omitted, if the text or previous appended blank line reached a trap position.
.sp N	-	$N = 1V$	B, v	Space vertically in <i>either</i> direction. If N is negative, the motion is <i>backward</i> (upward) and is limited to the distance to the top of the page. Forward (downward) motion is truncated to the distance to the nearest trap. If the no-space mode is on, no spacing occurs (see ns , and rs below).
.sv N	-	$N = 1V$	v	Save a contiguous vertical block of size N . If the distance to the next trap is greater than N , N vertical space is output. No-space mode has <i>no</i> effect. If this distance is less than N , no vertical space is immediately output, but N is remembered for later

			output (see os). Subsequent sv requests will overwrite any still remembered <i>N</i> .
.os		-	Output saved vertical space. No-space mode has <i>no</i> effect.
		-	Used to finally output a block of vertical space requested by an earlier sv request.
.ns	space	-	D No-space mode turned on. When on, the no-space mode inhibits sp requests and bp requests <i>without</i> a next page number. The no-space mode is turned off when a line of output occurs, or with rs .
.rs	space	-	D Restore spacing. The no-space mode is turned off.
	Blank text line.	-	B Causes a break and output of a blank line exactly like sp 1 .

5.9 LINE LENGTH AND INDENTING

The maximum line length for fill mode may be set with **ll**. The indent may be set with **in**; an indent applicable to *only* the *next* output line may be set with **ti**. The line length includes indent space but *not* page offset space. The line-length minus the indent is the basis for centering with **ce**. The effect of **ll**, **in**, or **ti** is delayed, if a partially collected line exists, until after that line is output. In fill mode the length of text on an output line is less than or equal to the line length minus the indent. The current line length and indent are available in registers **.l** and **.i** respectively. The length of three-part titles produced by **tl** is *independently* set by **lt**.

NROFF USER'S MANUAL

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES	EXPLANATION
.ll $\pm N$	6.5 in	previous	E,m	Line length is set to $\pm N$.
.in $\pm N$	$N=0$	previous	B,E,m	Indent is set to $\pm N$. The indent is prepended to each output line.
.ti $\pm N$	-	ignored	B,E,m	Temporary indent. The <i>next</i> output text line will be indented a distance $\pm N$ with respect to the current indent. The resulting total indent may not be negative. The current indent is not changed.

5.10 MACROS, STRINGS, DIVERSION, AND POSITION TRAPS

5.10.1 Macros and Strings

A *macro* is a named set of arbitrary *lines* that may be invoked by name or with a *trap*. A *string* is a named string of *characters*, *not* including a newline character, that may be interpolated by name at any point. Request, macro, and string names share the *same* name list. Macro and string names may be one or two characters long and may usurp previously defined request, macro, or string names. Any of these entities may be renamed with **rn** or removed with **rm**. Macros are created by **de** and **di**, and appended to by **am** and **da**; **di** and **da** cause normal output to be stored in a macro. Strings are created by **ds** and appended to by **as**. A macro is invoked in the same way as a request; a control line beginning **.xx** will interpolate the contents of macro **xx**. The remainder of the line may contain up to nine *arguments*. The strings **x** and **xx** are interpolated at any desired point with ***x** and ***(xx** respectively. String references and macro invocations may be nested.

5.10.2 Copy Mode Input Interpretation

During the definition and extension of strings and macros (not by diversion) the input is read in *copy mode*. The input is copied without interpretation *except* that:

- The contents of number registers indicated by \n are interpolated.
- Strings indicated by * are interpolated.
- Arguments indicated by \\$ are interpolated.
- Concealed newlines indicated by \newline are eliminated.
- Comments indicated by \" are eliminated.
- \t and \a are interpreted as ASCII horizontal tab and SOH respectively.
- \\ is interpreted as \.
- \. is interpreted as ":".

These interpretations can be suppressed by prepending a \. For example, since \\ maps into a \, \\n will copy as \n which will be interpreted as a number register indicator when the macro or string is reread.

5.10.3 Arguments

When a macro is invoked by name, the remainder of the line is taken to contain up to nine arguments. The argument separator is the space character, and arguments may be surrounded by double-quotes to permit imbedded space characters. Pairs of double-quotes may be imbedded in double-quoted arguments to represent a single double-quote. If the desired arguments won't fit on a line, a concealed newline may be used to continue on the next line.

When a macro is invoked the *input level* is *pushed down* and any arguments available at the previous level become unavailable until the macro is completely read and the previous level is restored. A macro's own arguments can be interpolated at *any* point within the macro with \\$N, which interpolates the Nth argument ($1 \leq N \leq 9$). If an invoked argument doesn't exist, a null string results. For example, the macro xx may be defined by

NROFF USER'S MANUAL

```
.de xx      \begin definition
Today is \$1 the \$2.
..          \end definition
```

and called by

```
.xx Monday 14th
```

to produce the text

Today is Monday the 14th.

Note that the `\$` was concealed in the definition with a prepended `\`. The number of currently available arguments is in the `.$` register.

No arguments are available at the top (non-macro) level in this implementation. Because string referencing is implemented as a input-level push down, no arguments are available from *within* a string. No arguments are available within a trap-invoked macro.

Arguments are copied in *copy mode* onto a stack where they are available for reference. The mechanism does not allow an argument to contain a direct reference to a *long* string (interpolated at copy time) and it is advisable to conceal string references (with an extra `\`) to delay interpolation until argument reference time.

5.10.4 Diversions

Processed output may be diverted into a macro for purposes such as footnote processing or determining the horizontal and vertical size of some text for conditional changing of pages or columns. A single diversion trap may be set at a specified vertical position. The number registers `dn` and `dl` respectively contain the vertical and horizontal size of the most recently ended diversion. Processed text that is diverted into a macro retains the vertical size of each of its lines when reread in *nofill* mode regardless of the current *V*. Constant-spaced (`cs`) or emboldened (`bd`) text that is diverted can be reread correctly only if these modes are again or still in effect at reread time. One way to do this is to imbed in the diversion the appropriate `cs` or `bd` requests with the *transparent* mechanism.

Diversions may be nested and certain parameters and registers are associated with the current diversion level (the top non-diversion level may be thought of as the 0th diversion level). These are the diversion trap and associated macro, no-space mode, the internally-saved marked place (see **mk** and **rt**), the current vertical place (**.d** register), the current high-water text base-line (**.h** register), and the current diversion name (**.z** register).

5.10.5 Traps

Three types of trap mechanisms are available—a page trap, a diversion trap, and an input-line-count trap. Macro-invocation traps may be planted using **wh** at any page position including the top. This trap position may be changed using **ch**. Trap positions at or below the bottom of the page have no effect unless or until moved to within the page or rendered effective by an increase in page length. Two traps may be planted at the *same* position only by first planting them at different positions and then moving one of the traps; the first planted trap will conceal the second unless and until the first one is moved. If the first one is moved back, it again conceals the second trap. The macro associated with a page trap is automatically invoked when a line of text is output whose vertical size *reaches* or *sweeps past* the trap position. Reaching the bottom of a page springs the top-of-page trap, if any, provided there is a next page. The distance to the next trap position is available in the **.t** register; if there are no traps between the current position and the bottom of the page, the distance returned is the distance to the page bottom.

A macro-invocation trap effective in the current diversion may be planted using **dt**. The **.t** register works in a diversion; if there is no subsequent trap a *large* distance is returned. For a description of input-line-count traps, see **it** below.

REQUEST	INITIAL	IF NO	ARGUMENT	NOTES	EXPLANATION
FORM	VALUE				
.de <i>xx yy</i>	-		<i>.yy = ..</i>	-	Define or redefine the macro <i>xx</i> . The contents of the macro begin on the next input line. Input lines are copied in <i>copy mode</i> until the definition is terminated by a line beginning with <i>.yy</i> , whereupon the macro <i>yy</i> is called. In the absence of <i>yy</i> , the

NROFF USER'S MANUAL

			definition is terminated by a line beginning with “..”. A macro may contain de requests provided the terminating macros differ or the contained definition terminator is concealed. “..” can be concealed as \\.. which will copy as \.. and be reread as “..”.	
.am <i>xx yy</i>	-	.yy=..	Append to macro (append version of de).	
.ds <i>xx string</i>	-	ignored	Define a string <i>xx</i> containing <i>string</i> . Any initial double-quote in <i>string</i> is stripped off to permit initial blanks.	
.as <i>xx string</i>	-	ignored	Append <i>string</i> to string <i>xx</i> (append version of ds).	
.rm <i>xx</i>	-	ignored	Remove request, macro, or string. The name <i>xx</i> is removed from the name list and any related storage space is freed. Subsequent references will have no effect.	
.rn <i>xx yy</i>	-	ignored	Rename request, macro, or string <i>xx</i> to <i>yy</i> . If <i>yy</i> exists, it is first removed.	
.di <i>xx</i>	-	end	D	Divert output to macro <i>xx</i> . Normal text processing occurs during diversion except that page offsetting is not done. The diversion ends when the request di or da is encountered without an argument; extraneous requests of this type should not appear when nested diversions are being used.
.da <i>xx</i>	-	end	D	Divert, appending to <i>xx</i> (append version of di).
.wh <i>N xx</i>	-	-	v	Install a trap to invoke <i>xx</i> at page position <i>N</i> ; a <i>negative N</i> will be interpreted with respect to

				the page <i>bottom</i> . Any macro previously planted at <i>N</i> is replaced by <i>xx</i> . A zero <i>N</i> refers to the <i>top</i> of a page. In the absence of <i>xx</i> , the first found trap at <i>N</i> , if any, is removed.
.ch <i>xx N</i>	-	-	v	Change the trap position for macro <i>xx</i> to be <i>N</i> . In the absence of <i>N</i> , the trap, if any, is removed.
.dt <i>N xx</i>	-	off	D,v	Install a diversion trap at position <i>N</i> in the <i>current</i> diversion to invoke macro <i>xx</i> . Another dt will redefine the diversion trap. If no arguments are given, the diversion trap is removed.
.it <i>N xx</i>	-	off	E	Set an input-line-count trap to invoke the macro <i>xx</i> after <i>N</i> lines of <i>text</i> input have been read (control or request lines don't count). The text may be in-line text or text interpolated by inline or trap-invoked macros.
.em <i>xx</i>	none	none	-	The macro <i>xx</i> will be invoked when all input has ended. The effect is the same as if the contents of <i>xx</i> had been at the end of the last file processed.

5.11 NUMBER REGISTERS

A variety of parameters are available to the user as predefined, named *number registers* (see Summary and Index). In addition, the user may define his own named registers. Register names are one or two characters long and *do not* conflict with request, macro, or string names. Except for certain predefined read-only registers, a number register can be read, written, automatically incremented or decremented, and interpolated into the input in a variety of formats. One common use of user-defined registers is to automatically number sections,

NROFF USER'S MANUAL

paragraphs, lines, etc. A number register may be used any time numerical input is expected or desired and may be used in numerical *expressions*.

Number registers are created and modified using **nr**, which specifies the name, numerical value, and the auto-increment size. Registers are also modified, if accessed with an auto-incrementing sequence. If the registers *x* and *xx* both contain *N* and have the auto-increment size *M*, the following access sequences have the effect shown:

Sequence	Effect on Register	Value Interpolated
\nx	none	<i>N</i>
\n(xx	none	<i>N</i>
\n+x	<i>x</i> incremented by <i>M</i>	<i>N+M</i>
\n-x	<i>x</i> decremented by <i>M</i>	<i>N-M</i>
\n+(xx	<i>xx</i> incremented by <i>M</i>	<i>N+M</i>
\n-(xx	<i>xx</i> decremented by <i>M</i>	<i>N-M</i>

When interpolated, a number register is converted to decimal (default), decimal with leading zeros, lower-case Roman, upper-case Roman, lower-case sequential alphabetic, or upper-case sequential alphabetic according to the format specified by **af**.

REQUEST	INITIAL FORM	IF NO VALUE	ARGUMENT	NOTES	EXPLANATION
.nr <i>R</i> $\pm N$ <i>M</i>	-			u	The number register <i>R</i> is assigned the value $\pm N$ with respect to the previous value, if any. The increment for auto-incrementing is set to <i>M</i> .
.af <i>R</i> <i>c</i>	arabic	-	-	-	Assign format <i>c</i> to register <i>R</i> . The available formats are:

Format	Numbering Sequence
1	0,1,2,3,4,5,...
001	000,001,002,003,004,005,...
i	0,i,ii,iii,iv,v,...
I	0,I,II,III,IV,V,...
a	0,a,b,c,...,z,aa,ab,...,zz,aaa,...
A	0,A,B,C,...,Z,AA,AB,...,ZZ,AAA,...

An arabic format having N digits specifies a field width of N digits (example 2 above). The read-only registers and the *width* function are always arabic.

.rr <i>R</i>	-	ignored	-	Remove register <i>R</i> . If many registers are being created dynamically, it may become necessary to remove no longer used registers to recapture internal storage space for newer registers.
--------------	---	---------	---	---

5.12 TABS, LEADERS, AND FIELDS

5.12.1 Tabs and Leaders

The ASCII horizontal tab character and the ASCII SOH (hereafter known as the *leader* character) can both be used to generate either horizontal motion or a string of repeated characters. The length of the generated entity is governed by internal *tab stops* specifiable with **ta**. The default difference is that tabs generate motion and leaders generate a string of periods; **tc** and **lc** offer the choice of repeated character or motion. There are three types of internal tab stops—*left* adjusting, *right* adjusting, and *centering*. In the following table: *D* is the distance from the current position on the *input* line (where a tab or leader was found) to the next tab stop; *next-string* consists of the input characters following the tab (or leader) up to the next tab (or leader) or end of line; and *W* is the width of *next-string*.

NROFF USER'S MANUAL

Tab type	Length of motion or repeated characters	Location of <i>next-string</i>
Left	D	Following D
Right	$D - W$	Right adjusted within D
Centered	$D - W/2$	Centered on right end of D

The length of generated motion is allowed to be negative, but that of a repeated character string cannot be. Repeated character strings contain an integer number of characters, and any residual distance is prepended as motion. Tabs or leaders found after the last tab stop are ignored, but may be used as *next-string* terminators.

Tabs and leaders are not interpreted in *copy mode*. `\t` and `\a` always generate a non-interpreted tab and leader respectively, and are equivalent to actual tabs and leaders in *copy mode*.

5.12.2 Fields

A *field* is contained between a *pair* of *field delimiter* characters, and consists of sub-strings separated by *padding indicator* characters. The field length is the distance on the *input* line from the position where the field begins to the next tab stop. The difference between the total length of all the sub-strings and the field length is incorporated as horizontal padding space that is divided among the indicated padding places. The incorporated padding is allowed to be negative. For example, if the field delimiter is # and the padding indicator is ^, `#^xxx^right #` specifies a right-adjusted string with the string `xxx` centered in the remaining space.

REQUEST	INITIAL	IF NO		
FORM	VALUE	ARGUMENT	NOTES	EXPLANATION
<code>.ta Nt ...</code>	<code>0.8; 0.5in</code>	<code>none</code>	<code>E,m</code>	Set tab stops and types. $t = \mathbf{R}$, right adjusting; $t = \mathbf{C}$, centering; t absent, left adjusting. Nroff every 0.8in. The stop values are separated by spaces, and a value preceded by + is treated as an increment to the previous stop value.

.tc <i>c</i>	none	none	E	The tab repetition character becomes <i>c</i> , or is removed specifying motion.
.lc <i>c</i>	.	none	E	The leader repetition character becomes <i>c</i> , or is removed specifying motion.
.fc <i>a b</i>	off	off	-	The field delimiter is set to <i>a</i> ; the padding indicator is set to the <i>space</i> character or to <i>b</i> , if given. In the absence of arguments the field mechanism is turned off.

5.13 INPUT/OUTPUT CONVENTIONS AND CHARACTER TRANSLATIONS

5.13.1 Input Character Translations

The ASCII control characters horizontal tab, SOH, and backspace are discussed elsewhere. The newline delimits input lines. In addition, STX, ETX, ENQ, ACK, and BEL are accepted, and may be used as delimiters or translated into a graphic with tr. All others are ignored.

The *escape* character \ introduces *escape sequences*—causes the following character to mean another character, or to indicate some function. A complete list of such sequences is given in the Summary and Index. \ should not be confused with the ASCII control character ESC of the same name. The escape character \ can be input with the sequence \\|. The escape character can be changed with ec, and all that has been said about the default \ becomes true for the new escape character. \e can be used to print whatever the current escape character is. If necessary or convenient, the escape mechanism may be turned off with eo, and restored with ec.

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES	EXPLANATION
.ec <i>c</i>	\	\	-	Set escape character to \, or to <i>c</i> , if given.
.eo	on	-	-	Turn escape mechanism off.

NROFF USER'S MANUAL

5.13.2 Ligatures

Five ligatures are available in the NR character set — **fi**, **fl**, **ff**, **ffi**, and **ffl**. They may be input by **\(fi**, **\(fl**, **\(ff**, **\(Fi**, and **\(Fl** respectively. The ligature mode is normally on in Troff, and *automatically* invokes ligatures during input.

REQUEST	INITIAL	IF NO		
FORM	VALUE	ARGUMENT	NOTES	EXPLANATION
.lg <i>N</i>	off	on	-	Ligature mode is turned on if <i>N</i> is absent or non-zero, and turned off if <i>N</i> =0. If <i>N</i> =2, only the two-character ligatures are automatically invoked.

5.13.3 Backspacing, Underlining, Overstriking, etc.

Unless in *copy mode*, the ASCII backspace character is replaced by a backward horizontal motion having the width of the space character.

Nroff automatically underlines characters in the *underline* font, specifiable with **uf**. In addition to **ft** and **\fF**, the underline font may be selected by **ul** and **cu**. Underlining is restricted to an output-device-dependent subset of *reasonable* characters.

REQUEST	INITIAL	IF NO		
FORM	VALUE	ARGUMENT	NOTES	EXPLANATION
.ul <i>N</i>	off	<i>N</i> =1	E	Underline in Nroff the next <i>N</i> input text lines. Actually, switch to <i>underline</i> font, saving the current font for later restoration; <i>other</i> font changes within the span of a ul will take effect, but the restoration will undo the last change. Output generated by tl is affected by the font change, but does <i>not</i> decrement <i>N</i> . If <i>N</i> >1, there is the risk that a trap interpolated macro may provide text lines within the span;

				environment switching can prevent this.
.cu <i>N</i>	off	<i>N</i> =1	E	A variant of ul that causes <i>every</i> character to be underlined in Nroff.
.uf <i>F</i>	Italic	Italic	-	Underline font set to <i>F</i> . In Nroff, <i>F</i> may <i>not</i> be on position 1 (initially Roman).

5.13.4 Control Characters

Both the control character `.` and the *no-break* control character `'` may be changed, if desired. Such a change must be compatible with the design of any macros used in the span of the change, and particularly of any trap-invoked macros.

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES	EXPLANATION
.cc <i>c</i>	.	.	E	The basic control character is set to <i>c</i> , or reset to <code>"."</code> .
.c2 <i>c</i>	'	'	E	The <i>nobreak</i> control character is set to <i>c</i> , or reset to <code>"'"</code> .

5.13.5 Output translation

One character can be made a stand-in for another character using **tr**. All text processing (e. g. character comparisons) takes place character. The graphic translation occurs at the moment of output (including diversion).

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES	EXPLANATION
.tr <i>abcd....</i>	none	-	O	Translate <i>a</i> into <i>b</i> , <i>c</i> into <i>d</i> , etc. If an odd number of characters is given, the last one will be mapped into the space character. To be consistent, a particular translation must stay in effect from <i>input</i> to <i>output</i> time.

NROFF USER'S MANUAL

5.13.6 Transparent Throughput

An input line beginning with a `\!` is read in *copy mode* and *transparently* output (without the initial `\!`); the text processor is otherwise unaware of the line's presence. This mechanism may be used to pass control information to a post-processor or to imbed control lines in a macro created by a diversion.

5.13.7 Comments and Concealed Newlines

An uncomfortably long input line that must stay one line (e. g. a string definition, or nofilled text) can be split into many physical lines by ending all but the last one with the escape `\`. The sequence `\(newline)` is *always* ignored—except in a comment. Comments may be imbedded at the *end* of any line by prefacing them with `\"`. The newline at the end of a comment cannot be concealed. A line beginning with `\"` will appear as a blank line and behave like `.sp 1`; a comment can be on a line by itself by beginning the line with `.\"`.

5.14 LOCAL MOTIONS AND THE WIDTH FUNCTION

5.14.1 Local Motions

The functions `\v'N'` and `\h'N'` can be used for *local* vertical and horizontal motion respectively. The distance *N* may be negative; the *positive* directions are *rightward* and *downward*. A *local* motion is one contained *within* a line. To avoid unexpected vertical dislocations, it is necessary that the *net* vertical local motion within a word in filled text and otherwise within a line balance to zero. The above and certain other escape sequences providing local motion are summarized in the following table.

Vertical Local Motion	Effect in Nroff	Horizontal Local Motion	Effect in Local Motion
<code>\v'N'</code>	Move distance <i>N</i>	<code>\h'N'</code>	Move distance <i>N</i>
<code>\u</code>	$\frac{1}{2}$ line up	<code>\(space)</code>	Unpaddable space-size space
<code>\d</code>	$\frac{1}{2}$ line down <code>\0</code>	Digit-size space	
<code>\r</code>	1 line up		

5.14.2 Width Function

The *width* function `\w'<string>'` generates the numerical width of *string* (in basic units). Size and font changes may be safely imbedded in *string*, and will not affect the current environment. For example, `.ti -\w'1.'` could be used to temporarily indent leftward a distance equal to the size of the string “1.”.

The width function also sets three number registers. The registers **st** and **sb** are set respectively to the highest and lowest extent of *string* relative to the baseline; then, for example, the total *height* of the string is **\n(stu - \n(sbu**. 0 means that all of the characters in *string* were short lower case characters without descenders (like e); 1 means that at least one character has a descender (like y); 2 means that at least one character is tall (like H); and 3 means that both tall characters and characters with descenders are present.

5.14.3 Mark Horizontal Place

The escape sequence `\kx` will cause the *current* horizontal position in the *input line* to be stored in register *x*. As an example, the construction `\kxword\h'|\nxu+2u`word` will embolden *word* by backing up to almost its beginning and overprinting it, resulting in *word*.

5.15 OVERSTRIKE, BRACKET, LINE-DRAWING, AND ZERO-WIDTH FUNCTIONS

5.15.1 Overstriking

Automatically centered overstriking of up to nine characters is provided by the *overstrike* function `\o'<string>'`. The characters in *string* overprinted with centers aligned; the total width is that of the widest character. *string* should *not* contain local vertical motion. An example is `\o'e\''` produces é.

5.15.2 Zero-width Characters

The function \zc will output *c* without spacing over it, and can be used to produce left-aligned overstruck combinations. As examples, \z\(\mathbf{ci}\)\(\mathbf{pl}\) will produce \oplus , and \(\mathbf{br}\z\(\mathbf{rn}\)\(\mathbf{ul}\)\(\mathbf{br}\) will produce the smallest possible constructed box \square .

NROFF USER'S MANUAL

5.15.3 Large Brackets

The Special Mathematical Font contains a number of bracket construction pieces (() [] { } | [] []) that can be combined into various bracket styles. The function `\b' string'` may be used to pile up vertically the characters in *string* (the first character on top and the last at the bottom); the characters are vertically separated by 1 line and the total pile is centered $\frac{1}{2}$ line above the current baseline. For example, `\b' \lc\lf'E\ \b' \rc\rf' \x' -0.5m' \x' 0.5m'` produces [E].

5.15.4 Line Drawing

The function `\l' Nc'` will draw a string of repeated *c*'s towards the right for a distance *N*. (`\l` is `\l`(lower case L). If *c* looks like a continuation of an expression for *N*, it may be insulated from *N* with a `\&`. If *c* is not specified, the underline character is used. If *N* is negative, a backward horizontal motion of size *N* is made *before* drawing the string. Any space resulting from *N*/(size of *c*) having a remainder is put at the beginning (left end) of the string. In the case of characters that are designed to be connected such as baseline-rule _, underrule __, and root-en '', the remainder space is covered by over-lapping. If *N* is less than the width of *c*, a single *c* is centered on a distance *N*. As an example, a macro to underscore a string can be written

```
.de us
\$1\l' 0\ul'
..
```

or one to draw a box around a string

```
.de bx
\br\ \$1\ \br\l' 0\rn'\l' 0\ul'
..
```

such that

```
.ul "underlined words"
```

and

```
.bx "words in a box"
```

yield underlined words and words in a box.

The function `\L'Nc'` will draw a vertical line consisting of the (optional) character *c* stacked vertically 1 line apart, with the first two characters overlapped, if necessary, to form a continuous line. The default character is the *box rule* | (`\(br`); the other suitable character is the *bold vertical* | (`\(bv`). The line is begun without any initial motion relative to the current base line. A positive *N* specifies a line drawn downward and a negative *N* specifies a line drawn upward. After the line is drawn *no* compensating motions are made; the instantaneous baseline is at the *end* of the line.

The horizontal and vertical line drawing functions may be used in combination to produce large boxes. The zero-width *box-rule* and the $\frac{1}{2}$ -em wide *underrule* were *designed* to form corners when using 1-em vertical spacings. For example the macro

```
.de eb
.sp -1  \"compensate for next automatic base-line spacing
.nf      \"avoid possibly overflowing word buffer
\h'-.5n'\L' |\\"nau-1'\l'\\"n(.lu+1n\ul'\L'-|\"nau+1'\l' 0u-.5n\ul'\"draw box
.fi
..
..
```

will draw a box around some text whose beginning vertical place was saved in number register *a* (e. g. using `.mk a`) as done for this paragraph.

5.16 HYPHENATION

The automatic hyphenation may be switched off and on. When switched on with `hy`, several variants may be set. A *hyphenation indicator* character may be imbedded in a word to specify desired hyphenation points, or may be prepended to suppress hyphenation. In addition, the user may specify a small exception word list.

Only words that consist of a central alphabetic string surrounded by (usually null) non-alphabetic strings are considered candidates for automatic hyphenation. Words that were input containing hyphens (minus), em-dashes (`\(em`), or hyphenation indicator characters—such as mother-in-law—are *always* subject to splitting after those characters, whether or not automatic hyphenation is on or off.

NROFF USER'S MANUAL

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES	EXPLANATION
.nh	hyphenate	-	E	Automatic hyphenation is turned off.
.hyN	on, $N=1$	on, $N=1$	E	Automatic hyphenation is turned on for $N \geq 1$, or off for $N = 0$. If $N = 2$, <i>last</i> lines (ones that will cause a trap) are not hyphenated. For $N = 4$ and 8 , the last and first two characters respectively of a word are not split off. These values are additive; i. e. $N = 14$ will invoke all three restrictions.
.hc c	\%	\%	E	Hyphenation indicator character is set to <i>c</i> or to the default \%. The indicator does not appear in the output.
.hw <i>word1 ...</i>	ignored	-		Specify hyphenation points in words with imbedded minus signs. Versions of a word with terminal <i>s</i> are implied; i.e. <i>dig-it</i> implies <i>dig-its</i> . This list is examined initially and after each sux stripping. The space available is small—about 128 characters.

5.17 THREE PART TITLES

The titling function **tl** provides for automatic placement of three fields at the left, center, and right of a line with a title-length specifiable with **lt**. **tl** may be used anywhere, and is independent of the normal text collecting process. A common use is in header and footer macros.

REQUEST	INITIAL	IF NO	ARGUMENT	NOTES	EXPLANATION
FORM	VALUE				
.tl 'left'center'right'		-	-	-	The strings <i>left</i> , <i>center</i> , and <i>right</i> are respectively left-adjusted, centered, and right-adjusted in the current title-length. Any of the strings may be empty, and overlapping is permitted. If the page-number character (initially %) is found within any of the fields it is replaced by the current page number having the format assigned to register %. Any character may be used as the string delimiter.
.pc <i>c</i>	%	off		-	The page number character is set to <i>c</i> , or removed. The page number register remains %.
.lt $\pm N$	6.5 in	previous	E,m		Length of title set to $\pm N$. The line-length and the title-length are <i>independent</i> . Indents do not apply to titles; page-offsets do.

5.18 OUTPUT LINE NUMBERING

Automatic sequence numbering of output lines may be requested with **nm**. When in effect, a three-digit, arabic number plus a digit-space is prepended to output text lines. The text lines are thus offset by four digit-spaces, and otherwise retain their line length; a reduction in line length may be desired to keep the right margin aligned with an earlier margin. Blank lines, other vertical spaces, and lines generated by **tl** are *not* numbered. Numbering can be temporarily suspended with **nn**, or with an **.nm** followed by a later **.nm + 0**. In addition, a line number indent *I*, and the number-text separation *S* may be specified in digit-spaces. Further, it can be specified that only those line numbers that are multiples of some number *M* are to be printed (the others will appear as blank number fields).

NROFF USER'S MANUAL

REQUEST	INITIAL	IF NO	
FORM	VALUE	ARGUMENT	NOTES
.nm $\pm NMSI$	off	E	Line number mode. If $\pm N$ is given, line numbering is turned on, and the next output line numbered is numbered $\pm N$. Default values are $M=1$, $S=1$, and $I=0$. Parameters corresponding to missing arguments are unaffected; a non-numeric argument is considered missing. In the absence of all arguments, numbering is turned off; the next line number is preserved for possible further use in number register In .
.nn N	-	$N=1$	E The next N text output lines are not numbered.

- 12 As an example, the paragraph portions of this section are numbered with $M=3$: **.nm 1 3** was placed at the beginning; **.nm** was placed at the end of the first paragraph; and **.nm +0** was placed in front of this paragraph; and
15 **.nm** finally placed at the end. Line lengths were also changed (by **\w'0000'u**) to keep the right side aligned. Another example is **.nm +5 5 x**
18 **3** which turns on numbering with the line number of the next line to be 5 greater than the last numbered line, with $M=5$, with spacing S untouched,
and with the indent I set to 3.

5.19 CONDITIONAL ACCEPTANCE OF INPUT

In the following, c is a one-character, built-in *condition* name, ! signifies *not*, N is a numerical expression, *string1* and *string2* are strings delimited by any non-blank, non-numeric character *not* in the strings, and *anything* represents what is conditionally accepted.

REQUEST FORM	INITIAL VALUE	IF NO ARGUMENT	NOTES	EXPLANATION
.if <i>c anything</i>	-	-		If condition <i>c</i> true, accept <i>anything</i> as input; in multi-line case use \{ <i>anything</i> \}.
.if ! <i>c anything</i>	-	-		If condition <i>c</i> false, accept <i>anything</i> .
.if <i>N anything</i>	-	u		If expression <i>N</i> > 0, accept <i>anything</i> .
.if ! <i>N anything</i>	-	u		If expression <i>N</i> ≤ 0, accept <i>anything</i> .
.if ' <i>string1</i> ' <i>string2</i> ' <i>anything</i>		-		If <i>string1</i> identical to <i>string2</i> , accept <i>anything</i> .
.if !' <i>string1</i> ' <i>string2</i> ' <i>anything</i>		-		If <i>string1</i> not identical to <i>string2</i> , accept <i>anything</i> .
.ie <i>c anything</i>	-	u		If portion of if-else; all above forms (like if).
.el <i>anything</i>	-	-		Else portion of if-else.

The built-in condition names are:

Condition Name	True If
o	Current page number is odd
e	Current page number is even
n	Formatter is Nroff

If the condition *c* is *true*, or if the number *N* is greater than zero, or if the strings compare identically (including motions and character size and font), *anything* is accepted as input. If a ! precedes the condition, number, or string comparison, the sense of the acceptance is reversed.

Any spaces between the condition and the beginning of *anything* are skipped over. The *anything* can be either a single input line (text, macro, or whatever) or a number of input lines. In the multi-line case, the first line must begin with a left delimiter \{ and the last line must end with a right delimiter \}.

NROFF USER'S MANUAL

The request **ie** (if-else) is identical to **if** except that the acceptance state is remembered. A subsequent and matching **el** (else) request then uses the reverse sense of that state. **ie - el** pairs may be nested.

Some examples are:

```
.if e .tl ' Even Page %'''
```

which outputs a title if the page number is even; and

```
.ie \n%>1 \{\  
'sp 0.5i  
.tl ' Page %'''  
'sp 1.2i \  
.el .sp 2.5i
```

which treats page 1 differently from other pages.

5.20 ENVIRONMENT SWITCHING

A number of the parameters that control the text processing are gathered together into an *environment*, which can be switched by the user. The environment parameters are those associated with requests noting E in their **NOTES** column; in addition, partially collected lines and words are in the environment. Everything else is global; examples are page-oriented parameters, diversion-oriented parameters, number registers, and macro and string definitions. All environments are initialized with default parameter values.

REQUEST	INITIAL	IF NO		
FORM	VALUE	ARGUMENT	NOTES	EXPLANATION
.ev N	N=0	previous	-	Environment switched to environment $0 \leq N \leq 2$. Switching is done in push-down fashion so that restoring a previous environment <i>must</i> be done with .ev rather than specific reference.

5.21 INSERTIONS FROM THE STANDARD INPUT

The input can be temporarily switched to the system *standard input* with **rd**, which will switch back when *two* newlines in a row are found (the *extra* blank line is not used). This mechanism is intended for insertions in form-letter-like documentation. On UNIX, the *standard input* can be the user's keyboard, a *pipe*, or a *file*.

REQUEST	INITIAL	IF NO		
FORM	VALUE	ARGUMENT	NOTES	EXPLANATION
.rd <i>prompt</i>	-	<i>prompt</i> = BEL	-	Read insertion from the standard input until two newlines in a row are found. If the standard input is the user's keyboard, <i>prompt</i> (or a BEL) is written onto the user's terminal. rd behaves like a macro, and arguments may be placed after <i>prompt</i> .
.ex	-	-	-	Exit from Nroff. Text processing is terminated exactly as if all input had ended.

If insertions are to be taken from the terminal keyboard *while* output is being printed on the terminal, the command line option **-q** will turn off the echoing of keyboard input and prompt only with BEL. The regular input and insertion input *cannot* simultaneously come from the standard input.

As an example, multiple copies of a form letter may be prepared by entering the insertions for all the copies in one file to be used as the standard input, and causing the file containing the letter to reinvoke itself using **nx**; the process would ultimately be ended by an **ex** in the insertion file.

5.22 INPUT/OUTPUT FILE SWITCHING

REQUEST	INITIAL	IF NO		EXPLANATION
FORM	VALUE	ARGUMENT	NOTES	
.so <i>filename</i>	-	-	-	Switch source file. The top input (file reading) level is switched to <i>filename</i> . The effect of an so encountered in a macro is not felt until the input level returns to the file level. When the new file ends, input is again taken from the original file. so 's may be nested.
.nx <i>filename</i>		end-of-file	-	Next file is <i>filename</i> . The current file is considered ended, and the input is immediately switched to <i>filename</i> .
.pi <i>program</i>	-	-	-	Pipe output to <i>program</i> . This request must occur <i>before</i> any printing occurs. No arguments are transmitted to <i>program</i> .

5.23 MISCELLANEOUS

REQUEST	INITIAL	IF NO		EXPLANATION
FORM	VALUE	ARGUMENT	NOTES	
.mc <i>c N</i>	-	off	E,m	Specifies that a <i>margin</i> character <i>c</i> appear a distance <i>N</i> to the right of the right margin after each non-empty text line (except those produced by tl). If the output line is too-long (as can happen in nofill mode) the character will be appended to the line. If <i>N</i> is not given, the previous <i>N</i> is used; the

.tm <i>string</i>	-	newline	-	initial <i>N</i> is 0.2 inches. The margin character used with this paragraph was a 12-point box-rule.
.ig <i>yy</i>	-	<i>yy=..</i>	-	After skipping initial blanks, <i>string</i> (rest of the line) is read in <i>copy mode</i> and written on the user's terminal.
.pm <i>t</i>	-	all	-	Ignore input lines. ig behaves exactly like de except that the input is discarded. The input is read in <i>copy mode</i> , and any auto-incremented registers will be affected.
.pm <i>t</i>	-	all	-	Print macros. The names and sizes of all of the defined macros and strings are printed on the user's terminal; if <i>t</i> is given, only the total of the sizes is printed. The sizes is given in <i>blocks</i> of 128 characters.
.fl	-	-	B	Flush output buffer. Used in interactive debugging to force output.

5.24 OUTPUT AND ERROR MESSAGES

The output from **tm**, **pm**, and the prompt from **rd**, as well as various *error* messages are written onto UNIX's *standard message* output. The latter is different from the *standard output*, where Nroff formatted output goes. By default, both are written onto the user's terminal, but they can be independently redirected.

Various *error* conditions may occur during the operation of Nroff. Certain less serious errors having only local impact do not cause processing to terminate. Two examples are *word overflow*, caused by a word that is too large to fit into the word buffer (in fill mode), and *line overflow*, caused by an output line that grew too large to fit in the line buffer; in both cases, a message is printed, the offending excess is discarded, and the affected word or line is marked at the point of truncation with an asterisk *. The philosophy is to continue processing, if possible, on the grounds that output useful for debugging may be

NROFF USER'S MANUAL

produced. If a serious error occurs, processing terminates, and an appropriate message is printed. Examples are the inability to create, read, or write files, and the exceeding of certain internal limits that make future output unlikely to be useful.

Table I**Input Naming Conventions for ', ` and —
and for Non-ASCII Special Characters****Non-ASCII characters and minus on the standard fonts.**

Char	Input Name	Character Name	Char	Input Name	Character Name
,	'	close quote	fi	\(fi	fi
'	`	open quote	fl	\(fl	fl
—	\(em	3/4 Em dash	ff	\(ff	ff
-	—	hyphen or	ffi	\(Fi	
-	\(hy	hyphen	ffl	\(Fl	ffl
—	\(—	current font minus	°	\(de	degree
●	\(bu	bullet	†	\(dg	dagger
□	\(sq	square	'	\(fm	foot mark
—	\(ru	rule	¢	\(ct	cent sign
¼	\(14	1/4	®	\(rg	registered
½	\(12	1/2	©	\(co	copyright
¾	\(34	3/4			

Non-ASCII characters and ', `_, +, -, =,

The ASCII characters @, #, “, ’, ‘, <, >, \, {, }, ~, ^, and _ exist *only* on the special font and are printed as a 1-em space if that font is not mounted. The following characters exist only on the special font except for the upper case Greek letter names followed by † which are mapped into upper case English letters in whatever font is mounted on font position one (default Times Roman). The special math plus, minus, and equals are provided to insulate the appearance of equations from the choice of standard fonts.

Char	Input	Character	Char	Input	Character
	Name	Name		Name	Name
+	\(pl	math plus	†	\(*U	Upsilon
-	\(mi	math minus	Φ	\(*F	Phi
=	\(eq	math equals	X	\(*X	Chi†
*	\(**	math star	Ψ	\(*Q	Psi
§	\(sc	section	Ω	\(*W	Omega
'	\(aa	acute accent	√	\(sr	square root
`	\(ga	grave accent	—	\(rn	root en extender
—	\(ul	underrule	≥	\(>=	>=
/	\(sl	slash (matching backslash)	≤	\(<=	<=
α	\(*a	alpha	≡	\(==	identically equal
β	\(*b	beta	≈	\(≈=	approx =
γ	\(*g	gamma	~	\(ap	approximates
δ	\(*d	delta	≠	\(!=	not equal
ε	\(*e	epsilon	→	\(->	right arrow
ζ	\(*z	zeta	←	\(<-	left arrow
η	\(*y	eta	↑	\(ua	up arrow
θ	\(*h	theta	↓	\(da	down arrow
ι	\(*i	iota	×	\(mu	multiply

NROFF USER'S MANUAL

\(*k	kappa	÷	\(di	divide
\(*l	lambda	±	\(+ -	plus-minus
\(*m	mu	∪	\(cu	cup (union)
\(*n	nu	∩	\(ca	cap (intersection)
\(*c	xi	⊂	\(sb	subset of
\(*o	omicron	⊃	\(sp	superset of
\(*p	pi	⊐	\(ib	improper subset
\(*r	rho	⊑	\(ip	improper superset
\(*s	sigma	∞	\(if	infinity
\(ts	terminal sigma	∂	\(pd	partial derivative
\(*t	tau	∇	\(gr	gradient
\(*u	upsilon	¬	\(no	not
\(*f	phi	∫	\(is	integral sign
\(*x	chi	∞	\(pt	proportional to
\(*q	psi	∅	\(es	empty set
\(*w	omega	€	\(mo	member of
\(*A	Alpha†		\(br	box vertical rule
\(*B	Beta†	‡	\(dd	double dagger
\(*G	Gamma	▀	\(rh	right hand
\(*D	Delta	▀	\(lh	left hand
\(*E	Epsilon†	▀	\(bs	Bell System logo
\(*Z	Zeta†	—	\(or	or
\(*Y	Eta†	○	\(ci	circle
\(*H	Theta	{	\(lt	left top of big curly bracket
\(*I	Iota†	{	\(lb	left bottom
\(*K	Kappa†	}	\(rt	right top
\(*L	Lambda	}	\(rb	right bot
\(*M	Mu†	{	\(lk	left center of big curly bracket
\(*N	Nu†	{	\(rk	right center of big curly bracket
\(*C	Xi	—	\(bv	bold vertical
\(*O	Omicron†	—	\(lf	left floor (left bottom of big square bracket)
\(*P	Pi	—	\(rf	right floor (right bottom)
\(*R	Rho†	—	\(lc	left ceiling (left top)
\(*S	Sigma	—	\(rc	right ceiling (right top)
\(*T	Tau†	—		

Chapter 6

NROFF TERMINAL DESCRIPTOR TABLE FORMAT

The **nroff** terminal driver tables describe the various characteristics of possible output terminals for nroff. The nroff flag **-Ttname** will cause the descriptor table **/usr/lib/term/tabtname** to be used.

The terminal table source is a C language file containing a single structure. It is compiled into a binary form through the **cc(1)** compiler (with the **-c** flag, since no linking needs to be done). The format of the table is as follows:

```
struct {
    int bset;
    int breset;
    int Hor;
    int Vert;
    int Newline;
    int Char;
    int Em;
    int Halfline;
    int Adj;
```

NROFF TERMINAL DESCRIPTOR TABLE FORMAT

```
char *twinit;
char *twrest;
char *twnl;
char *hlr;
char *hlf;
char *flr;
char *bdon;
char *bwoff;
char *ploton;
char *plotoff;
char *up;
char *down;
char *right;
char *left;
char *codetab[256-32];
int zzz;
};
```

1. The **bset** and **breset** numbers are used to set the terminal driver modes on the output terminal. When the terminal is opened, an **ioctl(2)** is done to set the terminal mode state. The state set is based on the previous mode and the **bset** and **breset** values:

new mode = old mode & ~breset | bset

The chapter VENIX PROGRAMMING in the *Programming Guide* describes these modes in detail. You should probably set the terminal to RAW mode to avoid unwanted conversions being done. When processing is finished, the original mode is restored.

2. **Hor** and **Vert** are the resolution of the terminal in basic units, where one inch equals 240 basic units. **Newline** is the length of one linefeed up or down; **Halffline** is the length of a halffline up or down. **Char**, **Em** and **Adj** are the width of a single character (internally, they have subtly different meaning to **nroff**, but in practice are all equivalent). All these numbers are in basic units.

NROFF TERMINAL DESCRIPTOR TABLE FORMAT

3. **Twinit** and **twrest** are character strings which are sent out to initialize and reset the output terminal, respectively, at the beginning and end of output. Any initialization codes particular to your output terminal can be placed here. **Twnl** is the string which is sent out at the end of every line, and normally contains a simple newline character.
4. **Hlr**, **hlf**, and **flr** are strings containing the control codes used to do a reverse half-linefeed, a forward half-linefeed, and a reverse full-linefeed respectively. If the terminal is not capable of doing half-linefeeds, a full-linefeed code is probably the next best thing; however, make sure that the **Halflne** value above reflects the actual height (which would be equal to **Newline** if the half-linefeed and full linefeed codes are identical). If your terminal can not do reverse linefeeds, you will probably wish to set those strings null.
5. The **bdon** and **bdooff** strings contain control codes used to turn on and off the bold mode, if any, on the output terminal. In **nroff** input, bold mode can be turned on and off with **.ft** or **\f** commands; the **bdon** and **bdooff** strings are then sent to the terminal to actually put bold modes on and off. Many terminals do not have this capability; however it is possible to construct a filter which can simulate bold face by overstriking the same character twice (or even better, striking a character, moving slightly to the right and hitting the same character again). If this is done, the **bdon** and **bdooff** strings can contain unique codes which the bold filter will intercept and use as signals to enter and exit the overstriking mode.
6. The **ploton** and **plotoff** strings are used to move into and out of graphics mode. In this mode, the **up**, **down**, **right** and **left** strings are then used to move one unit in the best resolution possible (as indicated by **Hor** and **Vert** above).
7. The **codetab** array contains a list of strings used to define each ASCII character and to simulate non-ASCII characters. A standard set is given in **/usr/lib/term/code.300** which will be adequate for most purposes, and a good starting point if you wish to custom-design your own.

The first character in each line is a character descriptor number. Bit 0200 is set if the character actually be underlined in underline mode (punctuation characters, for example, would normally not have this bit set). The bits masked by 0177 indicate the width of the character, which will normally be one unless fancy characters are used.

NROFF TERMINAL DESCRIPTOR TABLE FORMAT

The rest of the string is the printable portion. For standard ASCII characters, this will be simply the character used; for others, several characters (possibly with imbedded backspaces to cause them to be overstriked) can be used to create more complex characters.

8. To do fancier characters (as in Greek letters), it is possible to imbed plotting instructions within a character's **codetab** definition. Plotting instructions are in the form of '\nnn', and are octal values greater than 200 (to distinguish them from standard ASCII). The high three *bits* (masked by 0340) indicate motion in one of the four directions, corresponding to or'ing in the following octal values:

0340	up
0300	down
0240	left
0200	right

The lower five bits (masked by 034) specify the number of times to move in that direction, in the best possible resolution.

9. Note that plotting mode is turned on (string **ploton** sent) by the first plot code, and only turned off (**plotoff**) at the end of the string; for doing the plotting, control strings **up**, **down**, **right**, and **left** are written). Characters inserted between these plotting commands are sent to the output terminal, and handled by that terminal in plotting mode. For example, to construct the character π (indicated in nroff source as \(*p):

"\001\303'\203'\243\341'\203'\243\341~\201~\201~\201~\341\243",

As described above, the first \001 indicates that the character can not be underlined, and will be one em in width. The following codes are the character definition, and are interpreted as:

NROFF TERMINAL DESCRIPTOR TABLE FORMAT

\303	plotting mode on, mode down 3 units
'	write an apostrophe
\203	move right 3 units
'	write an apostrophe
\243	left 3 units
\341	up 1 unit

•
•
•

The \341\243 at the end moves the correct number of units back to the original position.

- Finally, zzz at the end of the table is a dummy variable.

CONTENTS

7

7.1 INTRODUCTION	7-1
7.2 INPUT COMMANDS	7-3
7.3 USAGE	7-11

Chapter 7

TABLE FORMATTING PROGRAM

7.1 INTRODUCTION

The **tbl** program is a document formatting preprocessor for the **nroff** formatter that makes fairly complex tables easy to specify and enter. Tables consist of columns which may be independently centered, right-adjusted, left-adjusted, or aligned by decimal points. Headings may be placed over single columns or groups of columns. A table entry may contain equations or consist of several rows of text. Horizontal or vertical lines may be drawn as desired in the table, and any table or element may be enclosed in a box.

A description of a table is translated by the **tbl** program into a list of **nroff** formatter requests that will produce the table. The **tbl** program isolates a portion of a job that it can successfully handle (text between the **.TS** and **.TE** delimiting macros) and leaves the remainder for other programs. Thus, **tbl** may be used with the equation formatting program (**neqn**) and/or various formatter layout macro packages without function duplication.

This chapter is divided into two parts and a reference list. The first part covers the rules for preparing **tbl** input; the second part shows several examples of how to use the program. The description of rules is precise but technical, so the novice user may find it easier to look over the examples first to see some common table arrangements and the instructions that created them. At the end of the chapter there is a list of **tbl** command characters and words which is useful as a reference guide.

TABLE FORMATTING PROGRAM

The input to **tbl** is text for a document, with tables preceded by a “.TS” (Table Start) command and followed by a “.TE” (Table End) command. **tbl** processes only the tables, generating **nroff** formatting commands. The “.TS” and “.TE” lines are copied, too, so that **nroff** page layout macros can use these lines to delimit and place tables as they see fit. In particular, any arguments on the “.TS” or “.TE” lines are copied but otherwise ignored, and may be used by document layout macro commands.

The format of the input is as follows:

```
text
.TS





```

where the format of each table is as follows:

```
.TS
options ;
format .
data
.TE
```

Each table is independent, and must contain formatting information followed by the data to be entered in the table. The formatting information, which describes the individual columns and rows of the table, may be preceded by a few options that affect the entire table.

7.2 Input Commands

As indicated above, a table contains, first, global options, then a format section describing the layout of the table entries, and then the data to be printed. The format and data are always required, but not the options. The various parts of the table are entered as follows:

1. OPTIONS. There may be a single line of options affecting the whole table. If present, this line must follow the .TS line immediately and must contain a list of option names separated by spaces, tabs, or commas, and must be terminated by a semicolon. The allowable options are:

center	— center the table (default is left-adjust);
expand	— make the table as wide as the current line length;
box	— enclose the table in a box;
allbox	— enclose each item in the table in a box;
doublebox	— enclose the table in two boxes;
tab (x)	— use <i>x</i> instead of tab to separate data items.
linesize (n)	— set lines or rules (e.g. from box) in <i>n</i> point type;
delim (xy)	— recognize <i>x</i> and <i>y</i> as the neqn delimiters.

The **tbl** program tries to keep boxed tables on one page by issuing appropriate “need” (.ne) commands. These requests are calculated from the number of lines in the tables, and if there are spacing commands embedded in the input, these requests may be inaccurate; use normal **nroff** procedures, such as keep-release macros, in that case. The user who must have a multi-page boxed table should use macros designed for this purpose, as explained below under “Usage.”

2. FORMAT. The format section of the table specifies the layout of the columns. Each line in this section corresponds to one line of the table (except that the last line corresponds to all following lines up to the next .T&, if any — see below), and each line contains a key-letter for each column of the table. It is good practice to separate the key letters for each column by spaces or tabs. Each key-letter is one of the following:

TABLE FORMATTING PROGRAM

- L or l** to indicate a left-adjusted column entry;
- R or r** to indicate a right-adjusted column entry;
- C or c** to indicate a centered column entry;
- N or n** to indicate a numerical column entry, to be aligned with other numerical entries so that the units digits of numbers line up;
- A or a** to indicate an alphabetic subcolumn; all corresponding entries are aligned on the left, and positioned so that the widest is centered within the column;
- S or s** to indicate a spanned heading, i.e. to indicate that the entry from the previous column continues across this column (not allowed for the first column, obviously); or
- ~** to indicate a vertically spanned heading, i.e. to indicate that the entry from the previous row continues down through this row. (Not allowed for the first row of the table, obviously).

When numerical alignment is specified, a location for the decimal point is sought. The rightmost dot (.) adjacent to a digit is used as a decimal point; if there is no dot adjoining a digit, the rightmost digit is used as a units digit; if no alignment is indicated, the item is centered in the column. However, the special non-printing character string \& may be used to override unconditionally dots and digits, or to align alphabetic data; this string lines up where a dot normally would, and then disappears from the final output. In the example below, the items shown at the left will be aligned (in a numerical column) as shown on the right:

13	13
4.2	4.2
26.4.12	26.4.12
abc	abc
abc\&	abc
43\&3.22	433.22
749.12	749.12

TABLE FORMATTING PROGRAM

Note: If numerical data is used in the same column with wider **L** or **r** type table entries, the widest number is centered relative to the wider **L** or **r** items (**L** is used instead of **I** for readability; they have the same meaning as key-letters). Alignment within the numerical items is preserved. This is similar to the behavior of **a** type data, as explained above. However, alphabetic subcolumns (requested by the **a** key-letter) are always slightly indented relative to **L** items; if necessary, the column width is increased to force this. This is not true for **n** type entries.

Warning: the **n** and **a** items should not be used in the same column.

For readability, the key-letters describing each column should be separated by spaces. The end of the format section is indicated by a period. The layout of the key-letters in the format section resembles the layout of the actual data in the table. Thus a simple format might appear as:

```
c s s  
l n n .
```

which specifies a table of three columns. The first line of the table contains a heading centered across all three columns; each remaining line contains a left-adjusted item in the first column followed by two columns of numerical data. A sample table in this format might be:

Overall title		
Item-a	34.22	9.1
Item-b	12.65	.02
Items: c,d,e	23	5.8
Total	69.87	14.92

There are some additional features of the key-letter system:

Horizontal lines

A key-letter may be replaced by ‘_’ (underscore) to indicate a horizontal line in place of the corresponding column entry, or by ‘=’ to indicate a double horizontal line. If an adjacent column contains a horizontal line, or if there are vertical lines adjoining this column, this horizontal line is extended to

TABLE FORMATTING PROGRAM

meet the nearby lines. If any data entry is provided for this column, it is ignored and a warning message is printed.

Vertical lines

A vertical bar may be placed between column key-letters. This will cause a vertical line between the corresponding columns of the table. A vertical bar to the left of the first key-letter or to the right of the last one produces a line at the edge of the table. If two vertical bars appear between key-letters, a double vertical line is drawn.

Space between columns

A number may follow the key-letter. This indicates the amount of separation between this column and the next column. The number normally specifies the separation in "ens" (one en is about the width of the letter "n").* If the "expand" option is used, then these numbers are multiplied by a constant such that the table is as wide as the current line length. The default column separation number is 3. If the separation is changed the worst case (largest space requested) governs.

Vertical spanning

Normally, vertically spanned items extending over several rows of the table are centered in their vertical range. If a key-letter is followed by t or T, any corresponding vertically spanned item will begin at the top line of its range.

Font changes

A key-letter may be followed by a string containing a font name or number preceded by the letter f or F. This indicates that the corresponding column should be in a different font from the default font (usually Roman). All font names are one or two letters; a one-letter font name should be separated

* More precisely, an en is a number of points (1 point = 1/72 inch) equal to half the current type size.

TABLE FORMATTING PROGRAM

from whatever follows by a space or tab. The single letters **B**, **b**, **I**, and **i** are shorter synonyms for **fB** and **fl**. Font change commands given with the table entries override these specifications.

Point size changes

A key-letter may be followed by the letter **p** or **P** and a number to indicate the point size of the corresponding table entries. The number may be a signed digit, in which case it is taken as an increment or decrement from the current point size. If both a point size and a column separation value are given, one or more blanks must separate them.

Vertical spacing changes

A key-letter may be followed by the letter **v** or **V** and a number to indicate the vertical line spacing to be used within a multi-line corresponding table entry. The number may be a signed digit, in which case it is taken as an increment or decrement from the current vertical spacing. A column separation value must be separated by blanks or some other specification from a vertical spacing request. This request has no effect unless the corresponding table entry is a text block (see below).

Column width indication

A key-letter may be followed by the letter **w** or **W** and a width value in parentheses. This width is used as a minimum column width. If the largest element in the column is not as wide as the width value given after the **w**, the largest element is assumed to be that wide. If the largest element in the column is wider than the specified value, its width is used. The width is also used as a default line length for included text blocks. Normal **nroff** units can be used to scale the width value; if none are used, the default is **ens**. If the width specification is a unitless integer the parentheses may be omitted. If the width value is changed in a column, the last one given controls.

TABLE FORMATTING PROGRAM

Equal width columns

A key-letter may be followed by the letter **e** or **E** to indicate equal width columns. All columns whose key-letters are followed by **e** or **E** are made the same width. This permits the user to get a group of regularly spaced columns.

Note: The order of the above features is immaterial; they need not be separated by spaces, except as indicated above to avoid ambiguities involving point size and font changes. Thus a numerical column entry in italic font and 12 point type with a minimum width of 2.5 inches and separated by 6 ens from the next column could be specified as

np12w(2.5i)fI 6

Alternative notation

Instead of listing the format of successive lines of a table on consecutive lines of the format section, successive line formats may be given on the same line, separated by commas, so that the format for the example above might have been written:

c s s, l n n .

Default Column descriptors missing from the end of a format line are assumed to be **L**. The longest line in the format section, however, defines the number of columns in the table; extra columns in the data are ignored silently.

3. DATA. The data for the table are typed after the format. Normally, each table line is typed as one line of data. Very long input lines can be broken: any line whose last character is **** is combined with the following line (and the **** vanishes). The data for different columns (the table entries) are separated by tabs, or by whatever character has been specified in the option *tabs* option. There are a few special cases:

TABLE FORMATTING PROGRAM

nroff commands within tables

An input line beginning with a ‘.’ followed by anything but a number is assumed to be a command to *nroff* and is passed through unchanged, retaining its position in the table. So, for example, space within a table may be produced by “.sp” commands in the data.

Full width horizontal lines

An input line containing only the character “_” (underline) or = (equal sign) is taken to be a single or double line, respectively, extending the full width of the table.

Single column horizontal lines

An input table entry containing only the character “_” or = is taken to be a single or double line extending the full width of the column. Such lines are extended to meet horizontal or vertical lines adjoining this column. To obtain these characters explicitly in a column, either precede them by \& or follow them by a space before the usual tab or newline.

Short horizontal lines

An input table entry containing only the string “_” is taken to be a single line as wide as the contents of the column. It is not extended to meet adjoining lines.

Vertically spanned items

An input table entry containing only the character string \^ indicates that the table entry immediately above spans downward over this row. It is equivalent to a table format key-letter of ‘^’.

Text blocks

In order to include a block of text as a table entry, precede it by “T{” and follow it by “T}”. Thus the sequence

... T{
block of
text
T} ...

is the way to enter, as a single entry in the table, something

TABLE FORMATTING PROGRAM

that cannot conveniently be typed as a simple string between tabs. Note that the “T!” end delimiter must begin a line; additional columns of data may follow after a tab on the same line. If more than twenty or thirty text blocks are used in a table, various limits in the **nroff** program are likely to be exceeded, producing diagnostics such as ‘too many string/macro names’ or ‘too many number registers.’ Text blocks are pulled out from the table, processed separately by **nroff**, and replaced in the table as a solid block. If no line length is specified in the block of text itself, or in the table format, the default is to use $L \times C / (N + 1)$ where L is the current line length, C is the number of table columns spanned by the text, and N is the total number of columns in the table. The other parameters (point size, font, etc.) used in setting the block of text are those in effect at the beginning of the table (including the effect of the “.TS” macro) and any table format specifications of size, spacing and font, using the **p**, **v** and **f** modifiers to the column key-letters. Commands within the text block itself are also recognized, of course. However, **nroff** commands within the table data but not within the text block do not affect that block.

Warnings: Although any number of lines may be present in a table, only the first 200 lines are used in calculating the widths of the various columns. A multi-page table, of course, may be arranged as several single-page tables if this proves to be a problem. Other difficulties with formatting may arise because, in the calculation of column widths all table entries are assumed to be in the font and size being used when the “.TS” command was encountered, except for font and size changes indicated (a) in the table format section and (b) within the table data (as in the entry `\s+3\fIdata\fP\s0`). Therefore, although arbitrary **nroff** requests may be sprinkled in a table, care must be taken to avoid confusing the width calculations; use requests such as ‘.ps’ with care.

TABLE FORMATTING PROGRAM

4. ADDITIONAL COMMAND LINES. If the format of a table must be changed after many similar lines, as with sub-headings or summarizations, the “.T&” (table continue) command can be used to change column parameters. The outline of such a table input is:

```
.TS  
options ;  
format .  
data  
...  
.T&  
format .  
data  
.T&  
format .  
data  
.TE
```

as in some of the examples in the second section. Using this procedure, each table line can be close to its corresponding format line.

Warning: it is not possible to change the number of columns, the space between columns, the global options such as **box**, or the selection of columns to be made equal width.

7.3 Usage

On VENIX **tbl** can be run on a simple table with the command

```
tbl input-file | nroff
```

but for more complicated use, where there are several input files, and they contain equations and **ms** memorandum layout commands as well as tables, the normal command would be

```
tbl file-1 file-2 . . . | neqn | nroff -ms
```

and, of course, the usual options may be used on the **nroff** and **neqn** commands. See **nroff(1)** and **neqn(1)** in the *User Reference Manual*

TABLE FORMATTING PROGRAM

For the convenience of users employing line printers without adequate driving tables or post-filters, there is a special **-TX** command line option to **tbl** which produces output that does not have fractional line motions in it. The only other command line options recognized by **tbl** are **-ms** and **-mm** which are turned into commands to fetch the corresponding macro files; usually it is more convenient to place these arguments on the **nroff** part of the command line, but they are accepted by **tbl** as well.

Note that when **neqn** and **tbl** are used together on the same file **tbl** should be used first. If there are no equations within tables, either order works, but it is usually faster to run **tbl** first, since **neqn** normally produces a larger expansion of the input than **tbl**. However, if there are equations within tables (using the **delim** mechanism in **neqn**), **tbl** must be first or the output will be scrambled. Users must also beware of using equations in **n**-style columns; this is nearly always wrong, since **tbl** attempts to split numerical format items into two parts and this is not possible with equations. The user can defend against this by giving the **delim(xx)** table option; this prevents splitting of numerical columns within the delimiters. For example, if the **neqn** delimiters are **\$\$**, giving **delim(\$\$)** a numerical column such as "1245 \$+ - 16\$" will be divided after 1245, not after 16.

tbl limits tables to twenty columns; however, use of more than 16 numerical columns may fail because of limits in **nroff**, producing the 'too many number registers' message. **nroff** number registers used by **tbl** must be avoided by the user within tables; these include two-digit names from 31 to 99, and names of the forms #*x*, *x*+, *x*|, *x*, and *x*-, where *x* is any lower case letter. The names ##, #-, and #^ are also used in certain circumstances. To conserve number register names, the **n** and **a** formats share a register; hence the restriction above that they may not be used in the same column.

For aid in writing layout macros, **tbl** defines a number register **TW** which is the table width; it is defined by the time that the ".TE" macro is invoked and may be used in the expansion of that macro. More importantly, to assist in laying out multi-page boxed tables the macro **T#** is defined to produce the bottom lines and side lines of a boxed table, and then invoked at its end. By use of this macro in the page footer a multi-page table can be boxed. In particular, the **ms** macros can be used to print a multi-page boxed table with a repeated heading by giving the argument **H** to the ".TS" macro. If the table start macro is written

TABLE FORMATTING PROGRAM

.TS H

a line of the form

.TH

must be given in the table after any table heading (or at the start if none). Material up to the “.TH” is placed at the top of each page of table; the remaining lines in the table are placed on several pages as required. Note that this is not a feature of **tbl**, but of the **ms** layout macros.

Examples

Here are some examples illustrating features of **tbl**. The symbol ⠄ in the input represents a tab character.

TABLE FORMATTING PROGRAM

Input:

```
.TS  
box;  
c c c  
l l l.  
Language ① Authors ① Runs on
```

```
Fortran ① Many ① Almost anything  
PL/1 ① IBM ① 360/370  
C ① BTL ① 11/45,H6000,370  
BLISS ① Carnegie-Mellon ① PDP-10,11  
IDS ① Honeywell ① H6000  
Pascal ① Stanford ① 370  
.TE
```

Output:

Language	Authors	Runs on
Fortran	Many	Almost anything
PL/1	IBM	360/370
C	BTL	11/45,H6000,370
BLISS	Carnegie-Mellon	PDP-10,11
IDS	Honeywell	H6000
Pascal	Stanford	370

TABLE FORMATTING PROGRAM

Input:

```
.TS
allbox;
c s s
c c c
n n n.
AT&T Common Stock
Year Price Dividend
1971 41-54 $2.60
2 41-54 2.70
3 46-55 2.87
4 40-53 3.24
5 45-52 3.40
6 51-59 .95*
.TE
* (first quarter only)
```

Output:

AT&T Common Stock		
Year	Price	Dividend
1971	41-54	\$2.60
2	41-54	2.70
3	46-55	2.87
4	40-53	3.24
5	45-52	3.40
6	51-59	.95*

* (first quarter only)

TABLE FORMATTING PROGRAM

Input:

```
.TS
box;
c s s
c | c | c
l | l | n.
Major New York Bridges
=
Bridge ① Designer ① Length

—
Brooklyn ① J. A. Roebling ① 1595
Manhattan ① G. Lindenthal ① 1470
Williamsburg ① L. L. Buck ① 1600

—
Queensborough ① Palmer & ① 1182
    ① Hornbostel

—
    ① ① 1380
Triborough ① O. H. Ammann ① _
    ① ① 383

—
Bronx Whitestone ① O. H. Ammann ① 2300
Throgs Neck ① O. H. Ammann ① 1800

—
George Washington ① O. H. Ammann ① 3500
.TE
```

Output:

Major New York Bridges		
Bridge	Designer	Length
Brooklyn	J. A. Roebling	1595
Manhattan	G. Lindenthal	1470
Williamsburg	L. L. Buck	1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O. H. Ammann	1380
		383
Bronx Whitestone	O. H. Ammann	2300
Throgs Neck	O. H. Ammann	1800
George Washington	O. H. Ammann	3500

TABLE FORMATTING PROGRAM

Input:

```
.TS
c c
np -2 | n |
① Stack
① -
1 ① 46
① -
2 ① 23
① -
3 ① 15
① -
4 ① 6.5
① -
5 ① 2.1
① -
.TE
```

Output:

Stack	
1	46
2	23
3	15
4	6.5
5	2.1

Input:

```
.TS
box;
L L L
L L -
L L | LB
L L -
L L L.
january ① february ① march
april ① may
june ① july ① Months
august ① september
october ① november ① december
.TE
```

Output:

january	february	march
april	may	
june	july	
august	september	Months
october	november	december

TABLE FORMATTING PROGRAM

Input:

```
.TS
box;
cB s s s.
Composition of Foods

.T&
c | c s s
c | c s s
c | c | c | c.
Food ① Percent by Weight
^① -
^① Protein ① Fat ① Carbo-
^① ^① ^① hydrate

.T&
l | n | n | n.
Apples ① .4 ① .5 ① 13.0
Halibut ① 18.4 ① 5.2 ① ...
Lima beans ① 7.5 ① .8 ① 22.0
Milk ① 3.3 ① 4.0 ① 5.0
Mushrooms ① 3.5 ① .4 ① 6.0
Rye bread ① 9.0 ① .6 ① 52.7
.TE
```

Output:

Composition of Foods			
Food	Percent by Weight		
	Protein	Fat	Carbo-hydrate
Apples	.4	.5	13.0
Halibut	18.4	5.2	...
Lima beans	7.5	.8	22.0
Milk	3.3	4.0	5.0
Mushrooms	3.5	.4	6.0
Rye bread	9.0	.6	52.7

TABLE FORMATTING PROGRAM

Input:

```
.TS
allbox;
cfl s s
c cw(1i) cw(1i)
lp9 lp9 lp9.

New York Area Rocks
Era ① Formation ① Age (years)
Precambrian ① Reading Prong ① >1 billion
Paleozoic ① Manhattan Prong ① 400 million
Mesozoic ① T{
.na
Newark Basin, incl.
Stockton, Lockatong, and Brunswick
formations; also Watchungs
and Palisades.
T} ① 200 million
Cenozoic ① Coastal Plain ① T{
On Long Island 30,000 years;
Cretaceous sediments redeposited
by recent glaciation.
.ad
T}
.TE
```

Output:

<i>New York Area Rocks</i>		
Era	Formation	Age (years)
Precambrian	Reading Prong	>1 billion
Paleozoic	Manhattan Prong	400 million
Mesozoic	Newark Basin, incl. Stockton, Lockatong, and Brunswick for- mations; also Watchungs and Palisades.	200 million
Cenozoic	Coastal Plain	On Long Island 30,000 years; Cretaceous sedi- ments redepos- ited by recent glaciation.

TABLE FORMATTING PROGRAM

Input:

```
.EQ
delim $$
.EN

. . .

.TS
doublebox;
c c
l l.

Name ① Definition
.sp
.vs +2p
Gamma ① $GAMMA (z) = int sub 0 sup inf t sup {z-1} e sup -t dt$
Sine ① $sin (x) = 1 over 2i ( e sup ix - e sup -ix )$
Error ① $ roman erf (z) = 2 over sqrt pi int sub 0 sup z e sup {-t sup 2} dt$
Bessel ① $ J sub 0 (z) = 1 over pi int sub 0 sup pi cos ( z sin theta ) d theta $
Zeta ① $ zeta (s) = sum from k=1 to inf k sup -s ~( Re s > 1 )$
.vs -2p
.TE
```

Output:

Name	Definition
Gamma	$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt$
Sine	$\sin(x) = \frac{1}{2i} (e^{ix} - e^{-ix})$
Error	$\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$
Bessel	$J_0(z) = \frac{1}{\pi} \int_0^{\pi} \cos(z \sin \theta) d\theta$
Zeta	$\zeta(s) = \sum_{k=1}^{\infty} k^{-s}$ (Re $s > 1$)

TABLE FORMATTING PROGRAM

Input:

```
.TS
box, tab(:);
cb s s s s
cp -2 s s s s
c || c | c | c | c
c || c | c | c | c
r2 || n2 | n2 | n2 | n.
Readability of Text
Line Width and Leading for 10-Point Type
=
Line : Set : 1-Point : 2-Point : 4-Point
Width : Solid : Leading : Leading : Leading
_____
9 Pica : -9.3 : -6.0 : -5.3 : -7.1
14 Pica : -4.5 : -0.6 : -0.3 : -1.7
19 Pica : -5.0 : -5.1 : 0.0 : -2.0
31 Pica : -3.7 : -3.8 : -2.4 : -3.6
43 Pica : -9.1 : -9.0 : -5.9 : -8.8
.TE
```

Output:

Readability of Text				
Line Width and Leading for 10-Point Type				
Line Width	Set Solid	1-Point Leading	2-Point Leading	4-Point Leading
9 Pica	-9.3	-6.0	-5.3	-7.1
14 Pica	-4.5	-0.6	-0.3	-1.7
19 Pica	-5.0	-5.1	0.0	-2.0
31 Pica	-3.7	-3.8	-2.4	-3.6
43 Pica	-9.1	-9.0	-5.9	-8.8

TABLE FORMATTING PROGRAM

Input:

```
.TS
c s
cip-2 s
l n
a n.
Some London Transport Statistics
(Year 1964)
Railway route miles ① 244
Tube ① 66
Sub-surface ① 22
Surface ① 156
.sp .5
.T&
l r
a r.
Passenger traffic \- railway
Journeys ① 674 million
Average length ① 4.55 miles
Passenger miles ① 3,066 million
.T&
l r
a r.
Passenger traffic \- road
Journeys ① 2,252 million
Average length ① 2.26 miles
Passenger miles ① 5,094 million
.T&
l n
a n.
.sp .5
Vehicles ① 12,521
Railway motor cars ① 2,905
Railway trailer cars ① 1,269
Total railway ① 4,174
Omnibuses ① 8,347
```

TABLE FORMATTING PROGRAM

.T&	
l n	
a n.	
.sp .5	
Staff ① 73,739	
Administrative, etc. 5,582	
Civil engineering ① 5,134	
Electrical eng. ① 1,714	
Mech. eng. \- railway ① 4,310	
Mech. eng. \- road ① 9,152	
Railway operations ① 8,930	
Road operations ① 35,946	
Other ① 2,971	
.TE	
	Some London Transport Statistics
	<i>(Year 1964)</i>
Output:	
	Railway route miles 244
	Tube 66
	Sub-surface 22
	Surface 156
	Passenger traffic — railway
	Journeys 674 million
	Average length 4.55 miles
	Passenger miles 3,066 million
	Passenger traffic — road
	Journeys 2,252 million
	Average length 2.26 miles
	Passenger miles 5,094 million
	Vehicles 12,521
	Railway motor cars 2,905
	Railway trailer cars 1,269
	Total railway 4,174
	Omnibuses 8,347
	Staff 73,739
	Administrative, etc. 5,582
	Civil engineering 5,134
	Electrical eng. 1,714
	Mech. eng. — railway 4,310
	Mech. eng. — road 9,152
	Railway operations 8,930
	Road operations 35,946
	Other 2,971

TABLE FORMATTING PROGRAM

Input:

```
.ps 8
.vs 10p
.TS
center box;
c s s
c i s s
c c c
IB 1 n.
New Jersey Representatives
(Democrats)
.sp .5
Name ① Office address ① Phone
.sp .5
James J. Florio ① 23 S. White Horse Pike, Somerdale 08083 ① 609-627-8222
William J. Hughes ① 2920 Atlantic Ave., Atlantic City 08401 ① 609-345-4844
James J. Howard ① 801 Bangs Ave., Asbury Park 07712 ① 201-774-1600
Frank Thompson, Jr. ① 10 Rutgers Pl., Trenton 08618 ① 609-599-1619
Andrew Maguire ① 115 W. Passaic St., Rochelle Park 07662 ① 201-843-0240
Robert A. Roe ① U.S.P.O., 194 Ward St., Paterson 07510 ① 201-523-5152
Henry Helstoski ① 666 Paterson Ave., East Rutherford 07073 ① 201-939-9090
Peter W. Rodino, Jr. ① Suite 1435A, 970 Broad St., Newark 07102 ① 201-645-3213
Joseph G. Minish ① 308 Main St., Orange 07050 ① 201-645-6363
Helen S. Meyner ① 32 Bridge St., Lambertville 08530 ① 609-397-1830
Dominick V. Daniels ① 895 Bergen Ave., Jersey City 07306 ① 201-659-7700
Edward J. Patten ① Natl. Bank Bldg., Perth Amboy 08861 ① 201-826-4610
```

TABLE FORMATTING PROGRAM

.sp .5
.T&
ci s s
lB l n.
(Republicans)
.sp .5v
Millicent Fenwick ① 41 N. Bridge St., Somerville 08876 ① 201-722-8200
Edwin B. Forsythe ① 301 Mill St., Moorestown 08057 ① 609-235-6622
Matthew J. Rinaldo ① 1961 Morris Ave., Union 07083 ① 201-687-4235
.TE
.ps 10
.vs 12p

Output:

New Jersey Representatives (Democrats)		
Name	Office address	Phone
James J. Florio	23 S. White Horse Pike, Somerdale 08083	609-627-8222
William J. Hughes	2920 Atlantic Ave., Atlantic City 08401	609-345-4844
James J. Howard	801 Bangs Ave., Asbury Park 07712	201-774-1600
Frank Thompson, Jr.	10 Rutgers Pl., Trenton 08618	609-599-1619
Andrew Maguire	115 W. Passaic St., Rochelle Park 07662	201-843-0240
Robert A. Roe	U.S.P.O., 194 Ward St., Paterson 07510	201-523-5152
Henry Helstoski	666 Paterson Ave., East Rutherford 07073	201-939-9090
Peter W. Rodino, Jr.	Suite 1435A, 970 Broad St., Newark 07102	201-645-3213
Joseph G. Minish	308 Main St., Orange 07050	201-645-6363
Helen S. Meyner	32 Bridge St., Lambertville 08530	609-397-1830
Dominick V. Daniels	895 Bergen Ave., Jersey City 07306	201-659-7700
Edward J. Patten	Natl. Bank Bldg., Perth Amboy 08861	201-826-4610
(Republicans)		
Millicent Fenwick	41 N. Bridge St., Somerville 08876	201-722-8200
Edwin B. Forsythe	301 Mill St., Moorestown 08057	609-235-6622
Matthew J. Rinaldo	1961 Morris Ave., Union 07083	201-687-4235

This is a paragraph of normal text placed here only to indicate where the left and right margins are. In this way the reader can judge the appearance of centered tables or expanded tables, and observe how such tables are formatted.

TABLE FORMATTING PROGRAM

Input:

```
.TS  
expand;  
c s s s  
c c c c  
l l n n.
```

Bell Labs Locations

Name	Address	Area Code	Phone
Holmdel	Holmdel, N. J. 07733	201	949-3000
Murray Hill	Murray Hill, N. J. 07974	201	582-6377
Whippany	Whippany, N. J. 07981	201	386-3000
Indian Hill	Naperville, Illinois 60540	312	690-2000

```
.TE
```

Output:

Bell Labs Locations			
Name	Address	Area Code	Phone
Holmdel	Holmdel, N. J. 07733	201	949-3000
Murray Hill	Murray Hill, N. J. 07974	201	582-6377
Whippany	Whippany, N. J. 07981	201	386-3000
Indian Hill	Naperville, Illinois 60540	312	690-2000

TABLE FORMATTING PROGRAM

Input:

```
.TS
box;
cb  s  s  s
c | c | c  s
ltw(1i) | ltw(2i) | lp8 | lw(1.6i)p8.
Some Interesting Places
```

Name	Description	Practical Information
------	-------------	-----------------------

T{	American Museum of Natural History	
----	------------------------------------	--

T} (T T{		
----------	--	--

The collections fill 11.5 acres (Michelin) or 25 acres (MTA)		
of exhibition halls on four floors. There is a full-sized replica		
of a blue whale and the world's largest star sapphire (stolen in 1964).		

T} (T Hours (T 10-5, ex. Sun 11-5, Wed. to 9		
--	--	--

^ (T ^ (T Location (T T{		
--------------------------	--	--

Central Park West & 79th St.		
------------------------------	--	--

T}		
----	--	--

^ (T ^ (T Admission (T Donation: \$1.00 asked		
---	--	--

^ (T ^ (T Subway (T AA to 81st St.		
------------------------------------	--	--

^ (T ^ (T Telephone (T 212-873-4225		
-------------------------------------	--	--

Bronx Zoo (T T{		
-----------------	--	--

About a mile long and .6 mile wide, this is the largest zoo in America.		
---	--	--

A lion eats 18 pounds		
-----------------------	--	--

of meat a day while a sea lion eats 15 pounds of fish.		
--	--	--

T} (T Hours (T T{		
-------------------	--	--

10-4:30 winter, to 5:00 summer		
--------------------------------	--	--

T}		
----	--	--

^ (T ^ (T Location (T T{		
--------------------------	--	--

185th St. & Southern Blvd, the Bronx.		
---------------------------------------	--	--

T}		
----	--	--

^ (T ^ (T Admission (T \$1.00, but Tu,We,Th free		
--	--	--

^ (T ^ (T Subway (T 2, 5 to East Tremont Ave.		
---	--	--

^ (T ^ (T Telephone (T 212-933-1759		
-------------------------------------	--	--

Brooklyn Museum (T T{		
-----------------------	--	--

Five floors of galleries contain American and ancient art.		
--	--	--

There are American period rooms and architectural ornaments saved		
from wreckers, such as a classical figure from Pennsylvania Station.		

T} (T Hours (T Wed-Sat, 10-5, Sun 12-5		
--	--	--

^ (T ^ (T Location (T T{		
--------------------------	--	--

Eastern Parkway & Washington Ave., Brooklyn.		
--	--	--

T}		
----	--	--

^ (T ^ (T Admission (T Free		
-----------------------------	--	--

^ (T ^ (T Subway (T 2,3 to Eastern Parkway.		
---	--	--

^ (T ^ (T Telephone (T 212-638-5000		
-------------------------------------	--	--

T{		
----	--	--

New-York Historical Society		
-----------------------------	--	--

T} (T T{		
----------	--	--

All the original paintings for Audubon's		
--	--	--

.I		
----	--	--

Birds of America		
------------------	--	--

TABLE FORMATTING PROGRAM

.R

are here, as are exhibits of American decorative arts, New York history, Hudson River school paintings, carriages, and glass paperweights.

T} ① Hours ① T{

Tues – Fri & Sun, 1 – 5; Sat 10 – 5

T}

^\circ ① ^\circ ① Location ① T{

Central Park West & 77th St.

T}

^\circ ① ^\circ ① Admission ① Free

^\circ ① ^\circ ① Subway ① AA to 81st St.

^\circ ① ^\circ ① Telephone ① 212-873-3400

.TE

Output:

Some Interesting Places				
Name	Description	Practical Information		
<i>American Museum of Natural History</i>	The collections fill 11.5 acres (Michelin) or 25 acres (MTA) of exhibition halls on four floors. There is a full-sized replica of a blue whale and the world's largest star sapphire (stolen in 1964).	Hours Location Admission Subway Telephone	10 – 5, ex. Sun 11 – 5, Wed. 1 Central Park West & 79th St Donation: \$1.00 asked AA to 81st St. 212-873-4225	
<i>Bronx Zoo</i>	About a mile long and .6 mile wide, this is the largest zoo in America. A lion eats 18 pounds of meat a day while a sea lion eats 15 pounds of fish.	Hours Location Bronx. Admission Subway Telephone	10 – 4:30 winter, to 5:00 summer 185th St. & Southern Blvd, Bronx \$1.00, but Tu,We,Th free 2, 5 to East Tremont Ave. 212-933-1759	
<i>Brooklyn Museum</i>	Five floors of galleries contain American and ancient art. There are American period rooms and architectural ornaments saved from wreckers, such as a classical figure from Pennsylvania Station.	Hours Location Admission Subway Telephone	Wed – Sat, 10 – 5, Sun 12 – 5 Eastern Parkway & Washington Ave., Brooklyn. Free 2,3 to Eastern Parkway. 212-638-5000	
<i>New-York Historical Society</i>	All the original paintings for Audubon's <i>Birds of America</i> are here, as are exhibits of American decorative arts, New York history, Hudson River school paintings, carriages, and glass paperweights.	Hours Location Admission Subway Telephone	Tues – Fri & Sun, 1 – 5; Sat 10 – 5 Central Park West & 77th St. Free AA to 81st St. 212-873-3400	

7.4 List of Tbl Command Characters and Words

<i>Command</i>	<i>Meaning</i>	<i>Section</i>
a A	Alphabetic subcolumn	2
allbox	Draw box around all items	1
b B	Boldface item	2
box	Draw box around table	1
c C	Centered column	2
center	Center table in page	1
doublebox	Doubled box around table	1
e E	Equal width columns	2
expand	Make table full line width	1
f F	Font change	2
i I	Italic item	2
l L	Left adjusted column	2
n N	Numerical column	2
nnn	Column separation	2
p P	Point size change	2
r R	Right adjusted column	2
s S	Spanned item	2
t T	Vertical spanning at top	2
tab (x)	Change data separator character	1
T{T}	Text block	3
v V	Vertical spacing change	2
w W	Minimum width value	2
.xx	Included nroff command	3
 	Vertical line	2
 	Double vertical line	2
^	Vertical span	2
\^	Vertical span	3
=	Double horizontal line	2,3
—	Horizontal line	2,3
_	Short horizontal line	3
\Rx	Repeat character	3

CONTENTS

8.1 INTRODUCTION	8-1
8.2 USAGE	8-2
8.3 DISPLAYED EQUATIONS	8-2
8.4 INPUT SPACES	8-3
8.5 OUTPUT SPACES	8-4
8.6 SYMBOLS, SPECIAL NAMES, AND GREEK ALPHABET	8-4
8.7 SUBSCRIPTS AND SUPERSCRIPTS	8-5
8.8 BRACES	8-7
8.9 FRACTIONS	8-8
8.10 SQUARE ROOTS	8-9
8.11 SUMMATIONS, INTEGRALS, AND SIMILAR CONSTRUCTIONS	8-9
8.12 DIACRITICAL MARKS	8-10
8.13 QUOTED TEXT	8-11
8.14 ALIGNING EQUATIONS	8-12
8.15 BIG BRACKETS	8-13
8.16 PILES	8-14
8.17 MATRICES	8-15
8.18 IN-LINE EQUATIONS	8-16
8.19 DEFINITIONS	8-17
8.20 LOCAL MOTIONS	8-18
8.21 PRECEDENCES	8-19
8.22 TROUBLESHOOTING	8-21

Chapter 8

MATHEMATICS TYPESETTING PROGRAM

8.1 INTRODUCTION

Mathematics is known in the publishing trade as “penalty copy” because it is slower, more difficult, and more expensive to set in type than any other kind of copy normally occurring in books and journals.

One difficulty with mathematical text is the multiplicity of characters, sizes, and fonts. Typesetting such expressions by traditional methods is still essentially a manual operation.

A second difficulty is the two-dimensional character of mathematics. This is illustrated by the following example which shows line-drawing, built-up characters (such as braces and radicals), and a spectrum of positioning problems:

$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \dots}}}$$

The **neqn** software for typesetting mathematics has been designed to be easy to learn and to use by people who know neither mathematics nor typesetting. The language can be learned in an hour or so since it has few rules and few exceptions. The syntax of the language is specified by a small content-free grammar.

neqn can be used on devices which have forward and reverse half-line motions. The equations produced by using this type of device in conjunction with **neqn** will not be high-quality, since they cannot provide for a variety of characters, sizes, and fonts; however, this output is usually adequate for proofreading.

NEQN

8.2 USAGE

On the VENIX system, **neqn** is a preprocessor for the **nroff** formatting program. To set the mathematical text stored in your files, the following command is issued:

```
neqn files | nroff
```

The vertical bar connects or “pipes” the output of the **neqn** process to the input of the **nroff** process. Any **nroff** formatter options (like a macro package) are located following the **nroff** formatter part of the command. For example:

```
neqn files | nroff -ms
```

To use a specific terminal as the output device, the following command is used:

```
neqn files | nroff -Tx
```

where *x* is the terminal type you are using, such as 1620 or LA50.

The **neqn** program can be used with the **tbl** program for setting tables that contain mathematics. Use **tbl** before **neqn**, like this:

```
tbl files | neqn | nroff
```

8.3 DISPLAYED EQUATIONS

To tell **neqn** where a mathematical expression begins and ends, mark it with lines beginning **.EQ** and **.EN**. Thus if you type the lines

```
.EQ  
x = y + z  
.EN
```

your output will look like

```
 $x = y + z$ 
```

The **.EQ** and **.EN** are copied through untouched; they are not otherwise processed by **neqn**. This means that you have to take care of things like centering, numbering, and so on yourself. The most common way is to use the **nroff** macro package package ‘**-ms**’, which allows you to center, indent, left-justify and number equations.

With the ‘**-ms**’ package, equations are centered by default. To left-justify an equation, use **.EQ L** instead of **.EQ**. To indent it, use **.EQ I**. Any of these can be followed by an arbitrary “equation number” which will be placed at the right margin. For example, the input

```
.EQ I (3.1a)
x = f(y/2) + y/2
.EN
```

produces the output

$$x = f(y/2) + y/2$$

8.4 INPUT SPACES

Input is free-form. Spaces and newlines in the input are used to separate pieces of the input; they are not used to create space in the output. Thus, between **.EQ** and **.EN**, in the following examples:

x = y + z

and

x = y + z

and

**x = y
+ z**

all produce the same output

$$x = y + z$$

8.5 OUTPUT SPACES

Extra white space can be forced into the output by several characters of various sizes. A “~” gives a space equal to the normal word spacing in the text; a circumflex gives half this much, and a tab character spaces to the next tab stop. Spaces, tildes, circumflexes, and tabs also serve to delimit pieces of the input. For example:

x~ = ~y~ + ~z

produces

$x = y + z$

8.6 SYMBOLS, SPECIAL NAMES, AND GREEK ALPHABET

neqn knows some mathematical symbols, some mathematical names, and the Greek alphabet. For example,

x=2 pi int sin (omega t)dt

produces

$x = 2\pi \int \sin(\omega t) dt$

Here the spaces in the input are necessary to tell **neqn** that “int”, “pi”, “sin” and “omega” are separate entities that should get special treatment. The sin, digit 2, and parentheses are set in roman type instead of italic; pi and omega are made Greek; and int becomes the integral sign.

When in doubt, leave spaces around separate parts of the input. A very common error is to type

f(pi)

without leaving spaces on both sides of the pi. As a result, **neqn** does not recognize pi as a special word, and it appears as *f(pi)* instead of *f(π)*.

The only way **neqn** can deduce that some sequence of letters might be special is if that sequence is separated from the letters on either side of it. This can be done by surrounding a special word by ordinary spaces (or tabs or newlines), as we did in the previous section.

You can also make special words stand out by surrounding them with tildes or circumflexes:

```
x~ = ~2~pi~int~sin~(~omega~t~)~dt
```

is much the same as the last example, except that the tildes not only separate the magic words like sin, omega, and so on, but also add extra spaces, one space per tilde:

$$x = 2 \pi \int \sin (\omega t) dt$$

Special words can also be separated by braces { } and double quotes "...", which have special meanings that we will see soon.

8.7 SUBSCRIPTS AND SUPERSCRIPTS

Subscripts and superscripts are obtained with the words **sub** and **sup**.

```
x sup 2 + y sub k
```

gives

$$x^2 + y_k$$

neqn takes care of all the size changes and vertical motions needed to make the output look right. The words **sub** and **sup** must be surrounded by spaces. For example:

```
x sub2
```

will give you

NEQN

x sub2 instead of x₂

Furthermore, don't forget to leave a space (or a tilde, etc.) to mark the end of a subscript or superscript. A common error is to say something like

y = (x sup 2)+1

which causes

$$y = (x^2) + 1$$

instead of the intended

$$y = (x^2) + 1$$

Subscripted subscripts and superscripted superscripts also work:

x sub i sub 1

is

$$x_{i1}$$

A subscript and superscript on the same thing are printed one above the other if the subscript comes first:

x sub i sup 2

is

$$x_i^2$$

8.8 BRACES

Normally, the end of a subscript or superscript is marked simply by a blank (or tab or tilde, etc.) What if the subscript or superscript is something that has to be typed with blanks in it? In that case, you can use the braces { and } to mark the beginning and end of the subscript or superscript:

e sup {i omega t}

is

$$e^{i \omega t}$$

Rule: Braces can always be used to force **neqn** to treat something as a unit, or just to make your intent perfectly clear. Thus:

x sub {i sub 1} sup 2

is

$$x_i^2_1$$

with braces, but

x sub i sub 1 sup 2

is

$$x_i^2_f$$

which is rather different.

Braces can occur within braces if necessary:

e sup {i pi sup {rho + 1}}

is

$$e^{i \pi^\rho + 1}$$

NEQN

The general rule is that anywhere you could use some single thing like x , you can use an arbitrarily complicated thing if you enclose it in braces. **neqn** will look after all the details of positioning it and making it the right size.

In all cases, make sure you have the right number of braces. Leaving one out or adding an extra will cause **neqn** to complain bitterly.

Occasionally you will have to print braces. To do this, enclose them in double quotes, like “{”. Quoting is discussed in more detail in section 14.

8.9 FRACTIONS

To make a fraction, use the word **over**:

a + b over 2c = 1

gives

$$\frac{a+b}{2c} = 1$$

The line is made the right length and positioned automatically. Braces can be used to make clear what goes over what:

{alpha + beta} over {sin (x)}

is

$$\frac{\alpha+\beta}{\sin(x)}$$

What happens when there is both an **over** and a **sup** in the same expression? In such an apparently ambiguous case, **neqn** does the **sup** before the **over**, so

-b sup 2 over pi

is $\frac{-b^2}{\pi}$ instead of $-b^{\frac{2}{\pi}}$. The rules which decide which operation is done first in cases like this are summarized in section 2. When in doubt, however, use braces to make clear what goes with what.

8.10 SQUARE ROOTS

To draw a square root, use `sqrt`.

`sqrt a+b + 1 over sqrt {ax sup 2 +bx+c}`

is

$$\sqrt{a+b} + \frac{1}{\sqrt{ax^2+bx+c}}$$

Warning — square roots of tall quantities look lousy, because a root-sign big enough to cover the quantity is too dark and heavy:

`sqrt {a sup 2 over b sub 2}`

is

$$\sqrt{\frac{a^2}{b_2}}$$

Big square roots are generally better written as something to the power $\frac{1}{2}$:

$$(a^2/b_2)^{\frac{1}{2}}$$

which is

$$(a^2/b_2)^{\text{sup half}}$$

8.11 SUMMATIONS, INTEGRALS, AND SIMILAR CONSTRUCTIONS

Summations, integrals, and similar constructions are easy:

`sum from i=0 to {i= inf} x sup i`

produces

$$\sum_{i=0}^{i=\infty} x^i$$

NEQN

Notice that we used braces to indicate where the upper part $i=inf$ begins and ends. No braces were necessary for the lower part $i=0$, because it contained no blanks. The braces will never hurt, and if the “from” and “to” parts contain any blanks, you must use braces around them.

The from and to parts are both optional, but if both are used, they have to occur in that order.

Other useful characters can replace the sum in our example:

int prod union inter

become, respectively,

\int \prod \cup \cap

Since the thing before the “from” can be anything, even something in braces, from-to can often be used in unexpected ways:

lim from {n -> inf} x sub n = 0

is

$$\lim_{n \rightarrow \infty} x_n = 0$$

8.12 DIACRITICAL MARKS

Diacritical marks, a problem in traditional typesetting, are straightforward in neqn. There are several words:

x dot	\dot{x}
x dotdot	\ddot{x}
x hat	\hat{x}
x tilde	\tilde{x}
x vec	\vec{x}
x dyad	\overline{x}
x bar	\bar{x}
x under	\underline{x}

The diacritical mark is placed at the right height. The **bar** and **under** are made the right length for the entire construct, as in $x+y+z$ *bar*; other marks are centered.

8.13 QUOTED TEXT

Any input entirely within quotes ("...") is not subject to any of the font changes and spacing adjustments normally done by the equation setter. This provides a way to do your own spacing and adjusting if needed:

italic "sin(x)" + sin (x)

is

sin(x) + sin(x)

Quotes are also used to get braces and other **neqn** keywords printed:

"{ size alpha }"

is

{ *size alpha* }

and

roman "{ size alpha }"

is

{ *size alpha* }

NEQN

The construction “” is often used as a place-holder when grammatically **neqn** needs something, but you don’t actually want anything in your output. For example, to make ${}^2\text{He}$, you can’t just type:

```
sup 2 roman He
```

A **sup** has to be a superscript on something. Thus you must say

```
"" sup 2 roman He
```

To get a literal quote use “\\”.

8.14 ALIGNING EQUATIONS

Sometimes it’s necessary to line up a series of equations at some horizontal position, often at an equals sign. This is done with two operations called **mark** and **lineup**.

The word **mark** may appear once at any place in an equation. It remembers the horizontal position where it appeared. Successive equations can contain one occurrence of the word **lineup**. The place where **lineup** appears is made to line up with the place marked by the previous **mark** if at all possible. Thus, for example, you can say

```
.EQ I  
x+y mark = z  
.EN  
.EQ I  
x lineup = 1  
.EN
```

to produce

$$x + y = z$$

$$x = 1$$

When you use **neqn** and ‘**-ms**’, use either **.EQ I** or **.EQ L**. **mark** and **lineup** don’t work with centered equations. Also bear in mind that **mark** doesn’t look ahead;

```
x mark = 1
...
x+y lineup = z
```

isn’t going to work, because there isn’t room for the **x+y** part after the **mark** remembers where the **x** is.

8.15 BIG BRACKETS

To get big brackets [], braces { }, parentheses (), and bars | | around things, use the **left** and **right** commands:

```
left { a over b + 1 right }
~ = ~ left ( c over d right )
+ left [ e right ]
```

is

$$\left\{ \frac{a}{b} + 1 \right\} = \left[\frac{c}{d} \right] + [e]$$

The resulting brackets are made big enough to cover whatever they enclose. Other characters can be used besides these, but they are not likely to look very good. One exception is the **floor** and **ceiling** characters:

```
left floor x over y right floor
<= left ceiling a over b right ceiling
```

produces

$$\left[\frac{x}{y} \right] \leq \left[\frac{a}{b} \right]$$

NEQN

Several warnings about brackets are in order. First, braces are typically bigger than brackets and parentheses, because they are made up of three, five, seven, etc., pieces, while brackets can be made up of two, three, etc. Second, big left and right parentheses often look poor, because the character set is poorly designed.

The right part may be omitted: a “left something” need not have a corresponding “right something”. If the right part is omitted, put braces around the thing you want the left bracket to encompass. Otherwise, the resulting brackets may be too large.

If you want to omit the left part, things are more complicated, because technically you can’t have a right without a corresponding left. Instead you have to say

```
left "" ... right )
```

for example. The left “ ” means a “left nothing”. This satisfies the rules without hurting your output.

8.16 PILES

There is a general facility for making vertical piles of things; it comes in several flavors. For example:

```
A ~ = ~ left [  
    pile { a above b above c }  
    ~ ~ pile { x above y above z }  
right ]
```

will make

$$A = \begin{bmatrix} a & x \\ b & y \\ c & z \end{bmatrix}$$

The elements of the pile (there can be as many as you want) are centered one

above another, at the right height for most purposes. The keyword **above** is used to separate the pieces; braces are used around the entire list. The elements of a pile can be as complicated as needed, even containing more piles.

Three other forms of pile exist: **lpile** makes a pile with the elements left-justified; **rpile** makes a right-justified pile; and **cpile** makes a centered pile, just like **pile**. The vertical spacing between the pieces is somewhat larger for **l-**, **r-** and **c**piles than it is for ordinary piles.

```
roman sign (x)~ = ~
left {
    lpile {1 above 0 above -1}
    ~ ~ lpile
    {if ~x>0 above if ~x=0 above if ~x<0}
```

makes

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Notice the left brace without a matching right one.

8.17 MATRICES

It is also possible to make matrices. For example, to make a neat array like

$$\begin{array}{ll} x_i & x^2 \\ y_i & y^2 \end{array}$$

you have to type

```
matrix {
    ccol { x sub i above y sub i }
    ccol { x sup 2 above y sup 2 }
}
```

NEQN

This produces a matrix with two centered columns. The elements of the columns are then listed just as for a pile, each element separated by the word **above**. You can also use **lcol** or **rcol** to left or right adjust columns. Each column can be separately adjusted, and there can be as many columns as you like.

The reason for using a matrix instead of two adjacent piles, by the way, is that if the elements of the piles don't all have the same height, they won't line up properly. A matrix forces them to line up, because it looks at the entire structure before deciding what spacing to use.

A word of warning about matrices — each column must have the same number of elements in it. The world will end if you get this wrong.

8.18 IN-LINE EQUATIONS

In a mathematical document, it is necessary to follow mathematical conventions not just in display equations, but also in the body of the text, for example by making variable names like *x* italic. Although this could be done by surrounding the appropriate parts with **.EQ** and **.EN**, the continual repetition of **.EQ** and **.EN** is a nuisance. Furthermore, with ‘**-ms**’, **.EQ** and **.EN** imply a displayed equation.

neqn provides a shorthand for short in-line expressions. You can define two characters to mark the left and right ends of an in-line equation, and then type expressions right in the middle of text lines. To set both the left and right characters to dollar signs, for example, add to the beginning of your document the three lines

```
.EQ  
  delim $%$%  
.EN
```

Having done this, you can then say things like

```
Let %alpha sub i% be the primary variable,  
and let %beta% be zero.  
Then we can show that %x sub 1% is %>=0%.
```

This works as you might expect — spaces, newlines, and so on are significant in

the text, but not in the equation part itself. Multiple equations can occur in a single input line.

Enough room is left before and after a line that contains in-line expressions that something like $\sum_{i=1}^n x_i$ does not interfere with the lines surrounding it.

To turn off the delimiters,

```
.EQ
delim off
.EN
```

Warning: don't use braces, tildes, circumflexes, or double quotes as delimiters; chaos will result.

8.19 DEFINITIONS

neqn provides a facility so you can give a frequently-used string of characters a name, and thereafter just type the name instead of the whole string. For example, if the sequence

```
x sub i sub 1 + y sub i sub 1
```

appears repeatedly throughout a paper, you can save re-typing it each time by defining it like this:

```
define xy 'x sub i sub 1 + y sub i sub 1'
```

This makes *xy* a shorthand for whatever characters occur between the single quotes in the definition. You can use any character instead of quote to mark the ends of the definition, so long as it doesn't appear inside the definition.

Now you can use *xy* like this:

```
.EQ
f(x) = xy ...
.EN
```

and so on. Each occurrence of *xy* will expand into what it was defined as. Be

NEQN

careful to leave spaces or their equivalent around the name when you actually use it, so **neqn** will be able to identify it as special.

There are several things to watch out for. First, although definitions can use previous definitions, as in:

```
.EQ  
define xi ' x sub i '  
define xil ' xi sub 1 '  
.EN
```

don't define something in terms of itself. A favorite error is to say:

```
define X ' roman X '
```

This is a guaranteed disaster, since *X* is now defined in terms of itself. If you say

```
define X ' roman "X" '
```

however, the quotes protect the second *X*, and everything works fine.

neqn keywords can be redefined. You can make / mean **over** by saying

```
define / ' over '
```

or redefine **over** as / with

```
define over ' / '
```

8.20 LOCAL MOTIONS

Although **neqn** tries to get most things at the right place on the paper, it isn't perfect, and occasionally you will need to tune the output to make it just right. Small extra horizontal spaces can be obtained with tilde and circumflex. You can also say **backn** and **fwdn** to move small amounts horizontally. *n* is how far to move in 1/100's of an em (an em is about the width of the letter 'm'.) Thus **back50** moves back about half the width of an m. Similarly you can move

things up or down with **upn** and **down n**. As with **sub** or **sup**, the local motions affect the next thing in the input, and this can be something arbitrarily complicated if it is enclosed in braces.

8.21 PRECEDENCES

If you don't use braces, **neqn** will do operations in the order shown in this list.

*dyad vec under bar tilde hat dot dotdot
 fwd back down up
 fat roman italic bold size
 sub sup sqrt over
 from to*

These operations group to the left:

over sqrt left right

All others group to the right.

Digits, parentheses, brackets, punctuation marks, and these mathematical words are converted to Roman font when encountered:

**sin cos tan sinh cosh tanh arc
 max min lim log ln exp
 Re Im and if for det**

These character sequences are recognized and translated as shown.

NEQN

> =	\geq
< =	\leq
= =	\equiv
!=	\neq
+ -	$+ -$
- >	$- >$
< -	$< -$
<<	$<<$
>>	$>>$
inf	∞
partial	∂
half	$\frac{1}{2}$
prime	'
approx	\approx
nothing	
cdot	.
times	\times
del	∇
grad	∇
...	...
,...,	,
sum	Σ
int	\int
prod	Π
union	\cup
inter	\cap

To obtain Greek letters, simply spell them out in whatever case you want:

DELTA	Δ	iota	ι
GAMMA	Γ	kappa	κ
LAMBDA	Λ	lambda	λ
OMEGA	Ω	mu	μ
PHI	Φ	nu	ν
PI	Π	omega	ω
PSI	Ψ	omicron	\circ
SIGMA	Σ	phi	ϕ
THETA	Θ	pi	π
UPSILON	Υ	psi	ψ
XI	Ξ	rho	ρ
alpha	α	sigma	σ
beta	β	tau	τ
chi	χ	theta	θ
delta	δ	upsilon	υ
epsilon	ϵ	xi	ξ
eta	η	zeta	ζ
gamma	γ		

8.22 TROUBLESHOOTING

If you make a mistake in an equation, like leaving out a brace (very common) or having one too many (very common) or having a `sup` with nothing before it (common), `neqn` will tell you with the message

syntax error between lines x and y, file z

where *x* and *y* are approximately the lines between which the trouble occurred, and *z* is the name of the file in question. The line numbers are approximate — look nearby as well. There are also self-explanatory messages that arise if you leave out a quote or try to run `neqn` on a non-existent file.

If you want to check a document before actually printing it

```
neqn files >/dev/null
```

NEQN

will throw away the output but print the messages.

If you use something like dollar signs as delimiters, it is easy to leave one out. This causes very strange troubles. The program **checkeq** checks for misplaced or missing dollar signs and similar troubles.

In-line equations can only be so big because of an internal buffer in **nroff**. If you get a message "word overflow", you have exceeded this limit. If you print the equation as a displayed equation this message will usually go away. The message "line overflow" indicates you have exceeded an even bigger buffer. The only cure for this is to break the equation into two separate ones.

On a related topic, **neqn** does not break equations by itself — you must split long equations up across multiple lines by yourself, marking each by a separate **.EQEN** sequence. **neqn** does warn about equations that are too long to fit on one line.

Printed in U.S.A.

AA-BM33A-TH