# Trnum – A Program To Number Figures

*Matt Bishop*

Research Institute for Advanced Computer Science
NASA Ames Research Center
Moffett Field, CA 94035

*ABSTRACT*

*Trnum* is a preprocessor for NROFF and TROFF text which enables users to label figures, displays, tables, and equations symbolically. This program then translates the labels into consecutive integers which may be printed in a variety of formats. In addition, prefixes and suffixes may be defined in such a way that they are automatically prepended or appended to the number. A number of different counters may be used, and the symbols may appear anywhere in the body of the paper. This document contains a tutorial as well as a detailed description of the program.

July 1986

# Trnum – A Program To Number Figures

*Matt Bishop*

Research Institute for Advanced Computer Science
NASA Ames Research Center
Moffett Field, CA 94035

## I. Introduction

A problem which authors of papers often face is that of numbering figures, tables, sections, equations, and the like† while the paper is being written. One often finds that a new display must be inserted between two others; as a result, the numbering of all such displays after the newly-inserted one is not correct. Tracking down and correcting the erroneous numbers is very time-consuming and painstaking. More often than not, one will overlook a number; the result may be that the reader will become confused.

*Trnum* alleviates this problem. The author labels his displays with a *pair* of names; when a new display is inserted, the numbers corresponding to the pairs following the insertion will be updated automatically. The user need not change any of the pairs in the text, because *trnum* numbers pairs in the order in which they occur. At no time does *trnum* generate commands targeted for the text processing program; rather, it generates the required labels itself. Thus it can be used with any text processing program, or indeed any program requiring items to be numbered.

The next three sections describe the mechanics of controlling *trnum*. The two sections after that describe how *trnum* is invoked under UNIX†, and some lessons learned

† These will be called *displays* throughout this paper.

† UNIX is a Trademark of Bell Laboratories.

from its development and use. The last section is a tutorial on using *trnum*, complete with examples using the text processors NROFF and TROFF [1].

## II. Overview

*Trnum* copies most text from its input files to the standard output; certain strings called *pairs* (which are described in the next section) are replaced by labels, and lines beginning with ‘‘.\" trnum:’’ allow the user to force certain actions to occur. These will be discussed separately.

*Trnum* maintains 101 *counters*, each of which begins at 1 and increases as new symbols are defined. The user may name these counters as he pleases. Each counter allows 2039 different symbols, the names of which the user chooses; a symbol may be defined in as many counters as the user likes. Each counter and symbol *pair* represents a different label; two different counters may have symbols with the same name, but the numbers associated with the two pairs need not be the same.

When a counter is created, certain attributes associated with that counter are set as indicated by that pair. These attributes are the *prefix* (a string which is printed before the number associated with the pair), the *suffix* (a string which is printed after the number associated with the pair), and the *format* (a character indicating how the number associated with the pair is to be printed). Whenever this counter is referenced, unless new values of these attributes are given in the pair referencing that counter, the values used will be those associated with the counter.

**III. Pairs**

A counter and symbol is referred to by a sequence of characters called a *pair*, which has the following format:

<[*prefix*+][*counter*.][*symbol*][:*format*][−*suffix*]>

The brackets enclose parts of the pair associated with one another. In detail, the parts are:

*prefix* This is a sequence of characters terminated by "+". It is prepended to the number associated with the counter and symbol. The default is the null string.

*counter* This is a sequence of alphanumeric characters terminated by "." (period). It names the counter to which the symbol refers. The default is the counter "default".

*symbol* This is a sequence of alphanumeric characters which names the symbol. There is no default.

*format* This character indicates in what form the number associated with the counter and symbol is to be printed, and it is preceded by ":" (colon). 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 (see the entry for **001** in the table.) Note the formats are the same as for the macro

"`.af`" in NROFF and TROFF. The default format is format **1** (Arabic numbers in the form of line 1 of the above table.)

*suffix* This is a sequence of characters preceded by "`−`" and terminated by "`>`". It is appended to the label number associated with the counter and symbol. The default suffix is the null string.

Notice that any character except "`+`", "`.`", "`:`", "`−`", "`<`", "`>`", and "`%`" may appear in the prefix and the suffix; to put of these characters (called *control characters*) in the prefix or suffix, precede it with the *quote* character "`%`". (For example, "`%%`" in the prefix will set the prefix to "`%`".) Only letters (both upper and lower case), digits, and the underscore character may appear in counter and symbol names. In addition, the format string *must* be one of the ones described there (that is, "`i`", "`I`", "`a`", "`A`", or a string of 1 or more Arabic digits); erroneous formats are *not* caught. If a sequence of characters does not meet the description above, *trnum* will assume the sequence is not a pair and simply copy it to the output. In general, this is *trnum*'s approach to all input: if it is a pair, replace it – otherwise, copy it.

One of the symbol and counter names must be present; any of the other parts of the pair may be omitted. In all cases where a symbol name is present, the appropriate default associated with the counter is used when something is omitted. (If the counter name is omitted, the name "default" is used.) If the symbol name is omitted, but the counter name is present, the pair is copied to the output, but any counter referenced by it (the one named "default", if no name is given explicitly) is created and initialized. This is most useful when the default attributes of the counter are to be set differently than the values required the first time the counter is used.

Here are some examples of pairs:

<Figure +fig.fifo_queue:I–%.>

A counter is defined whenever it is first encountered, so if the counter ‘‘fig’’ has not been used before this pair occurs, this pair will define it, and set the default prefix, suffix, and format for that counter to the strings ‘‘Figure ’’, ‘‘.’’, and ‘‘I’’, respectively. (Notice that the blank in the prefix need not be quoted.) In this case, the text ‘‘Figure I.’’ would replace the pair. If the counter had been encountered before (for this example, suppose ‘‘fifo_queue’’ were the fourth symbol defined for that counter), *regardless* of the attributes associated with the counter ‘‘fig’’, the pair would be replaced by ‘‘Figure IV.’’, since any attributes named explicitly in the pair override the default attributes associated with the counter. Thus, to suppress the prefix and suffix and force Arabic format, give explicitly null attributes (for the above example,

<+fig.fifo_queue:–>

will suppress all default attributes and use the default, Arabic, format.)

<maintheorem>

This defines a symbol, ‘‘maintheorem’’, which is associated with the default counter. It will be printed as an Arabic numeral. If the pair

<maintheorem:i>

is encountered later on, the number associated with the symbol ‘‘maintheorem’’ and counter ‘‘default’’ will be printed as a lower-case Roman numeral.

**IV. Verbs**

Occasionally it is useful to be able to exercise more detailed control over counters and their component symbols. For example, when beginning a new chapter of a thesis,

certain counters, such as a subsection number counter, should be reset. *Trnum* offers these capabilities by means of *verbs*.

A verb is a command indicating that special action is to be taken. It is given on a line beginning with ‘‘.\" trnum:’’, immediately following the colon (blank spaces may be put between the colon and the verb.) Note these lines are treated as comment lines by NROFF and TROFF and so are ignored. Only enough of the verb to identify it uniquely need be given. Any *arguments* to the verb are placed after it. The verbs are:

*begin* Change the character indicating that a pair begins to the first character of the first argument to the verb; for example,

.\" trnum: begin (

indicates that pairs will begin with ‘‘(’’. Initially, this character is ‘‘<’’.

*control* This macro prints on the error output (usually the terminal) the control characters and verbs.

*define* This verb indicates the pairs named as arguments are to be defined. It is useful when pairs must be numbered in an order other than that in which they occur. If the symbol name in a pair is omitted, the counter is initialized but no symbol is entered.

*end* Change the character indicating that a pair ends to the first character of the first argument to the verb; for example,

.\" trnum: end )

indicates that pairs will end with ‘‘)’’. Initially, this character is ‘‘>’’.

*format* Change the character indicating the format begins to the first character of the first argument to the verb; for example,

.\" trnum: format #

indicates that the format description will begin with "#". Initially, this character is ":" (colon).

*off* This verb turns off the interpretation of pairs; *trnum* just copies the input to the output until interpretation is turned back on (see *on*, below.) It is useful when escaping character sequences which look like pairs but are not pairs, and which *trnum* would otherwise replace with numbers. (An alternative is to quote the beginning character of the pair.)

*on* This verb turns on the interpretation of pairs. See *off*, above.

*prefix* Change the character terminating the prefix of the first argument to the verb; for example,

.\" trnum: prefix ;

indicates that prefixes will end with ";". Initially, this character is "+".

*print* This macro prints to the error output (usually the terminal) the label associated with any pairs given as arguments. If the symbol name is omitted, the labels of all symbols associated with the named counter are printed; if the counter name is omitted, the labels of all symbols with that name (in all counters) are printed; and if both the symbol and counter names are omitted, the labels of all defined pairs are printed.

*quote* Change the quote character to the first character of the first argument to the verb; for example,

.\" trnum: quote \

indicates that the quote character will be "\". Initially, this character is

“%”.

*reset* This macro indicates the pairs named as arguments are to be reset; it is identical to undefining the pair, then defining it again, and is included as a convenience.

*separator* Change the character separating the counter name from the symbol name to the first character of the first argument to the verb; for example,

```
.\" trnum: separator ,
```

indicates that the counter name and the symbol name will be separated by “,”. Initially, this character is “.”.

*suffix* Change the character indicating the suffix begins to the first character of the first argument to the verb; for example,

```
.\" trnum: suffix *
```

indicates that suffixes will begin with “*”. Initially, this character is “–”.

*terse* This turns off *verbose* mode.

*undefine* This verb indicates the pairs named as arguments are to be undefined. It deletes them from the appropriate counters. Pairs occurring as arguments are replaced by their definition before being undefined. If the symbol name in a pair is omitted, the counter is undefined; this deletes both the counter and all its symbols.

*verbose* This verb turns on *verbose* mode. In verbose mode, every pair encountered is printed; this is most useful for debugging input text. By default this mode is off; the command-line option **–v** will turn it on for all the input. This mode may be turned on with this verb, and turned off with the

verb *terse*.

Note that the control lines are copied to the output, but pairs occurring on them are not replaced. The verbs *define*, *undefine*, *reset*, and *print* may have more than one argument; but the verbs *begin*, *end*, *separator*, *format*, *prefix*, *quote*, and *suffix* ignore any text after the first character of the next word. If there is no nonblank word following those verbs, the appropriate character is turned off, so it cannot be matched; since no two control characters may be the same, this is necessary to change one control character to another. For example, if the control character beginning pairs is "<", the control character ending pairs is ">", and they are to be interchanged, the appropriate sequence of commands is:

```
.\" trnum: begin
.\" trnum: end   <
.\" trnum: begin >
```

As another example, the following changes the characters which begin and end pairs and separate counter names from symbol names:

```
.\" trnum: begin {
.\" trnum: end   }
.\" trnum: separ ,
```

After these three lines, {a1,b2} will be recognized as a pair; however, <a1.b2> will not.

**V. Use**

To run *trnum* under UNIX, simply type

```
trnum
```

followed by a list of files. If no files are specified, or the file "–" is encountered, *trnum* reads from the standard input. Output is written on the standard output.

With the NROFF and TROFF text processors, *trnum* should come before any other preprocessor; for example,

trnum *file* ... | refer | pic | tbl | eqn | troff

(see [2], [3], [4], and [5].)

There are four options to *trnum.* The option **–C** causes *trnum* to accept input for an earlier version of that preprocessor; see the next section for details. For version 2 and later, the option **–I** prints the version number and date of *trnum* on the standard output; on version 1, this switch does not exist (and the program prints an error message.) The option **–v** prints pairs on the error output as they are encountered in the text; it is useful for making tables of symbols and counters. And the option **–s** causes *trnum* not to print control lines, which is necessary when using *trnum* with *TEX* [6].

## VI. History and Experience

*Trnum* was written in 1983 to number sections and equations in Ph.D. dissertations, and since then has been used at Purdue University, the University of California at Santa Barbara, and the Research Institute for Advanced Computer Science. From the experiences of the users, some lessons have become clear.

Because of the power of the NROFF and TROFF command language, it is entirely possible to implement a numbering facility using the number and string registers within those text processors, rather than writing a separate preprocessor. However, the resulting mechanism is neither so general nor so powerful as that provided by *trnum*. Specifically, the numbering is tied to NROFF and TROFF; it could be used with *TEX*, for example. Labels are restricted to two characters because NROFF and TROFF only permits one or two character register names; this limit hampers users in remembering what the appropriate

label is called, just as one or two character variable names confuse programmers [7]. Moreover, there is an internal limit on the number of registers available within NROFF and TROFF; as *trnum* does not use these registers, the user need not worry about these internal limits. All these advantages vindicate the decision to write *trnum* as a preprocessor.

There are a few visible differences between this version of *trnum* and the earlier version. In the earlier version, prefixes and suffixes could consist only of letters, digits, underscores, and quoted characters; because of the nature of NROFF and TROFF commands, this was changed. Also, the default quote character was '\' rather than '%'. This was changed because '\' is used to indicate to NROFF and TROFF the need for special actions, such as interpolating a special character or the value contained in a register. For example, to use the contents of the string register *Ab* as a prefix and the contents of the number register *Cd* as a suffix (both separated from the number by a space), one had to type

```
<\\*(Ab\\ +counter.symbol–\\ \\n(Cd>
```

to the first version of *trnum*. To later versions, one need type only

```
<\*(Ab +counter.symbol– \n(Cd>
```

because one need not worry about *trnum* stripping off any '\' characters. The '%' sign was chosen because it has no special meaning to NROFF and TROFF except when used in titles, and then only when escaped.

In version 1, verbs were placed on lines beginning with the NROFF and TROFF macro *.NU*. For example, to change the character separating counters and symbols to a comma, one used to write

```
.NU separator ,
```

rather than

.\" trnum: separator ,

(as one would now.) There seemed to be no good reason for using a NROFF and TROFF macro register when *trnum* was run as a preprocessor; more importantly, when the macro *.NU* was defined, NROFF and TROFF output was not what was expected. Lines beginning with the string ".\"" are comments in NROFF and TROFF and so will always be ignored. (With other programs, just use the **–s** option to suppress these lines.)

To enable documents written to use version 1 *trnum* to be run without changes, give the **–C** flag. Note that the new verbs will be recognized, but the quote character and verb line indicator will be what the earlier version expects. To determine the version being run, give the command

trnum –I

On the earlier version, this will produce the message

trnum: bad switch –I

and leave *trnum* waiting for input (just type your end-of-file symbol, usually control-D, to cause it to exit); on this (and future) versions, it will print a version number and date.

## VII. A Tutorial

Suppose you are typing a paper, and you decide that you want to number your tables. Pick a symbol; say, "format". Then, whenever you refer to this table, rather than saying "table number 1," you would say "table number <format>". *Trnum* will replace "<format>" with the appropriate number wherever it occurs. (If you left off the angle brackets "<...>", *trnum* would *not* replace anything. The angle brackets signal *trnum* to look at what they enclose; if it is a legal string (we'll get to what's legal in a bit), then,

and only then, does *trnum* do the replacement.

*Trnum* assigns a number to a symbol the *first* time it is seen; hence, you do have to be careful in what order you refer to labelled things. For example, in the input

```
The next tables give information about the attributes
which affect the definitions of counters.
Notice that table <defaults> contains the default
settings for the attributes in table <attributes>:
.TS
box, center, tab(;);
c s
cB2 | c
cB2 | l.
<attributes>. Attributes.
Name;Meaning
_
prefix;prepended to label
counter;counter name
symbol;symbol name
format;how to print label
suffix;appended to label
.TE
.TS
center, box, tab(;);
c s
cB2 | c
cB2 | c
cB2 | l.
<defaults>. Defaults
What;Default
\^;Values
_
prefix;\fInull string\fP
format;\fB1\fP
counter;\fBdefault\fP
suffix;\fInull string\fP
.TE
```

the label "<defaults>" will be replaced by **1**, and the label "<attributes>" by **2**, because that is the order the labels occur in the text:

```
The next tables give information about the attributes which affect the definitions of counters.
Notice that table 1 contains the default settings for the attributes in table 2:
```

| 2. Attributes. | |
|---|---|
| **Name** | Meaning |
| **prefix** | prepended to label |
| **counter** | counter name |
| **symbol** | symbol name |
| **format** | how to print label |
| **suffix** | appended to label |

| 1. Defaults | |
|---|---|
| **What** | Default Values |
| **prefix** | *null string* |
| **format** | **1** |
| **counter** | **default** |
| **suffix** | *null string* |

Unfortunately, it's also backwards – table 1 should precede table 2.

To allow this, *trnum* lets you define labels on command lines, which begin with ".\" trnum:". Notice that these are ignored by NROFF and TROFF since they are comments. So, to get the labels to come out right in the above example, just before the first line, insert the line

.\" trnum: define <attributes>

Then, since "<attributes>" came before "<defaults>", it will be defined as **1**, and the tables will be numbered in the order they appear.

The next tables give information about the attributes which affect the definitions of counters. Notice that table 2 contains the default settings for the attributes in table 1:

| 1. Attributes. | |
|---|---|
| **Name** | Meaning |
| **prefix** | prepended to label |
| **counter** | counter name |
| **symbol** | symbol name |
| **format** | how to print label |
| **suffix** | appended to label |

| 2. Defaults | |
|---|---|
| **What** | Default Values |
| **prefix** | *null string* |
| **format** | **1** |
| **counter** | **default** |
| **suffix** | *null string* |

Usually, you want to keep several counts going at the same time; for example, you rarely want to number an equation as 1, the next table as 2, the next equation as 3, and so forth. What you usually want is to number the first equation as 1, the first table as 1, the next equation as 2, and then refer to them as equation 1, table 1, and so forth. To enable you to do this, *trnum* has several different *counters*; each starts at 1. To name a particular counter, prefix the symbol with the counter name followed by a period ‘‘.’’. In the above example, if we wanted to use a counter named ‘‘table’’ to keep track of the tables, we would write ‘‘<table.attributes>’’ and ‘‘<table.defaults>’’ instead of ‘‘<attributes>’’ and ‘‘<defaults>’’, respectively. Like symbols, a counter is initialized when it is first encountered; unlike symbols, when a counter is initialized, it always starts at 1. You can define up to 101 different counters, and if you omit the counter name, a default counter is used. (This counter’s name, incidentally, is ‘‘default’’.)

Sometimes you would like to restart a counter at 1. There are two ways to do this. You could delete the counter, and then redefine it with

```
.\" trnum: undefine <countername.>
.\" trnum:   define <countername.>
```

Alternatively, you could say

```
.\" trnum: reset <countername.>
```

which does the same thing. Note the ‘‘.’’; you *have* to have it or *trnum* will think you mean the symbol *countername* and not the counter. Omitting the symbol causes the

action to affect the counter rather than a symbol. Also, the string is passed through *trnum*; unlike strings involving symbols, it is not replaced by anything. Be aware that when you reset or undefine a counter, it undefines all symbols for that counter too.

Incidentally, you can delete a symbol, too. Each counter can have 2039 symbols defined for it; this should be more than enough for most people. However, if it is not, to delete a symbol, say:

.\" trnum: undefine <*countername.symbolname*>

Resetting a symbol just undefines it, and then redefines it.

Normally, numbers are printed in Arabic format, without any leading zeroes. This is not always what you want; for instance, some people prefer that tables be numbered with Roman numerals. *Trnum* allows you to control how a counter is printed by a *format* field. The format field follows the symbol name, and is preceded by a colon (“:”). For example, to print the table labels as Roman numerals in the example above, we would write “<table.attributes:I>” and “<table.defaults:I>” (the “I” indicates capital Roman numerals. Other formats available are lower-case Roman numerals (the format indicator is “i”), capital letters (format indicator “A”), lower-case letters (format indicator “a”), and Arabic numerals with or without leading zeroes (indicate this by an Arabic number containing as many digits as you want printed, including leading zeroes.) The format indicator is an attribute of the *counter*, not of the symbol.

This last comment deserves explanation. When a counter is initialized, several attributes are created (one of which is the format indicator; we’ll deal with the others below.) The format specified in the label which creates the counter becomes the default format for that counter. It can be overridden for any label by specifying a format

explicitly in that label, but if no format indicator is given, the default one will be used.

```
.\" trnum: define <table.:I>
.\" trnum: define <table.attributes>
The next tables give information about the attributes
which affect the definitions of counters.
Notice that table <table.defaults> contains the default
settings for the attributes in table <table.attributes:00>:
.TS
box, center, tab(;);
c s
cB2 | c
cB2 | l.
<table.attributes>. Attributes.
Name;Meaning
_
prefix;prepended to label
counter;counter name
symbol;symbol name
format;how to print label
suffix;appended to label
.TE
.TS
center, box, tab(;);
c s
cB2 | c
cB2 | c
cB2 | l.
<table.defaults>. Defaults
What;Default
\^;Values
_
prefix;\fInull string\fP
format;\fB1\fP
counter;\fBdefault\fP
suffix;\fInull string\fP
.TE
```

In this text, all table numbers will be printed as Roman numerals, because when the counter ‘‘table’’ is created (by the first ‘‘.\" trnum:’’), its format indicator is ‘‘I’’. However, when the number for the label ‘‘<table.attributes:00>’’ is printed, since there is an explicit format indicator with the label (the ‘‘00’’) it will be printed as a two-digit Arabic numeral:

> The next tables give information about the attributes which affect the definitions of counters.
> Notice that table II contains the default settings for the attributes in table 01:

| I. Attributes. | |
|---|---|
| **Name** | Meaning |
| **prefix** | prepended to label |
| **counter** | counter name |
| **symbol** | symbol name |
| **format** | how to print label |
| **suffix** | appended to label |

| II. Defaults | |
|---|---|
| **What** | Default Values |
| **prefix** | *null string* |
| **format** | **1** |
| **counter** | **default** |
| **suffix** | *null string* |

Notice that only the label with the ‘‘00’’ format indicator given explicitly is affected.

There are two other attributes which may be set; a *prefix*, which is prepended to every label which uses that counter, and a *suffix*, which is appended to every label which uses that counter, The prefix precedes the counter name and is followed by a plus sign (‘‘+’’); the suffix follows the format indicator, and is preceded by a minus sign (‘‘−’’). As with the format indicator, the defaults may be explicitly overridden in any label.

Going back to our perennial example, notice in the two lines of text the table number is preceded by the word ‘‘table’’ and a blank. To get this prefix prepended automatically, we would define the counter by:

.\" trnum: define <table +table.:I>

Then, the sample text becomes:

```
.\" trnum: define <table +table.:I>
.\" trnum: define <table.attributes>
The next tables give information about the attributes
which affect the definitions of counters.
Notice that <table.defaults> contains the default
settings for the attributes in <table.attributes:00>:
.TS
box, center, tab(;);
c s
```

```
cB2 | c
cB2 | l.
<+table.attributes>. Attributes.
Name;Meaning
_
prefix;prepended to label
counter;counter name
symbol;symbol name
format;how to print label
suffix;appended to label
.TE
.TS
center, box, tab(;);
c s
cB2 | c
cB2 | c
cB2 | l.
<+table.defaults>. Defaults
What;Default
;Values
_
prefix;null string
format;1
counter;default
suffix;null string
.TE
```

(notice the explicitly null prefixes within the tables themselves) and prints as:

The next tables give information about the attributes which affect the definitions of counters. Notice that table II contains the default settings for the attributes in table 01:

I. Attributes.

| **Name** | Meaning |
|---|---|
| **prefix** | prepended to label |
| **counter** | counter name |
| **symbol** | symbol name |
| **format** | how to print label |
| **suffix** | appended to label |

II. Defaults

| **What** | Default Values |
|---|---|
| **prefix** | *null string* |
| **format** | **1** |
| **counter** | **default** |
| **suffix** | *null string* |

Suffixes are handled the same way. Note that within the same text, the table numbers in the body of the tables end with a period. However, since a period separates the

counter and symbol, it is a control character, and hence must be escaped. Thus, we must write

.\" trnum: define <table +table.:I–%.>

and the text would look like:

```
.\" trnum: define <table +table.:I–%.>
.\" trnum: define <table.attributes>
The next tables give information about the attributes
which affect the definitions of counters.
Notice that <table.defaults–> contains the default
settings for the attributes in <table.attributes:00–>:
.TS
box, center, tab(;);
c s
cB2 | c
cB2 | l.
<+table.attributes> Attributes.
Name;Meaning
_
prefix;prepended to label
counter;counter name
symbol;symbol name
format;how to print label
suffix;appended to label
.TE
.TS
center, box, tab(;);
c s
cB2 | c
cB2 | c
cB2 | l.
<+table.defaults> Defaults
What;Default
\^;Values
_
prefix;\f2null string\fP
format;\f31\fP
counter;\f3default\fP
suffix;\f2null string\fP
```

As a final example, here is the text of a proof of the irrationality of *e*, the base of the natural logarithms [8]:

```
.\" trnum: define <(+equation.:1–)>
Recall
```

```
.EQ <equation.edef>
e ~=~ sum from k=0 to inf {1 over k!} ~ .
.EN
Choose
.EQ <equation.e1>
s sub n ~=~ sum from k=0 to n {1 over k!} ~ .
.EN
By <equation.e1> and <equation.edef>, we see
.EQ <equation.e2>
0 ~<~ e ~-~ s sub n ~ < ~ { 1 over n!n }~ .
.EN
Assume
.I e
is rational;
then $e ~=~ p over q$,
where
.I p
and
.I q
are positive integers.
By <equation.e2>,
.EQ <equation.e3>
0 ~<~ q!~(e~-~s sub q ) ~<~{1 over q} ~ .
.EN
By our assumption, $q!e$ is an integer.
Since by <equation.e1>,
.EQ <equation.e4>
q! s sub q ~=~ q! ( 1 ~+~ 1 ~+~ {1 over 2!} ~+~ ... ~+~ {1 over q!} )
.EN
is an integer,
we see that $q! ( e ~-~ s sub q )$ is an integer.
As $q ~ >=~ 1$,
<equation.e3> implies there is an integer between 0 and 1, contradiction.
```

which produces:

Recall

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} . \tag{1}$$

Choose

$$s_n = \sum_{k=0}^{n} \frac{1}{k!} . \tag{2}$$

By (2) and (1), we see

$$0 < e - s_n < \frac{1}{n!n} . \tag{3}$$

Assume $e$ is rational; then $e = \frac{p}{q}$, where $p$ and $q$ are positive integers. By (3),

$$0 < q! \, (e - s_q) < \frac{1}{q} . \tag{4}$$

By our assumption, $q!e$ is an integer. Since by (2),

$$q!s_q = q!(1 + 1 + \frac{1}{2!} + \cdots + \frac{1}{q!}) \tag{5}$$

is an integer, we see that $q!(e - s_q)$ is an integer. As $q \geq 1$, (4) implies there is an integer between 0 and 1, contradiction.

This covers the basics of *trnum*; the first sections of this guide explain some other, less often used, features. In case you ever want to see what labels you have defined, the line

.\" trnum: print <>

will cause all defined labels to be printed on the standard error. You can restrict the printing to one counter by naming it, as:

.\" trnum: print <*countername*.>

The other forms of this statement are less useful, and are documented in the second section.

The subject of naming things deserves mention. The characters which may be used to name symbols and counters are all letters (both cases), all digits, and underscore (“_”). The characters which may be used in prefixes and suffixes are any except the control characters “<”, “+”, “.”, “:”, “–”, “>”, and “%”, and these may be used by prefixing them with the quote character “%”. This is done specifically to allow string registers to be named in the prefix or suffix strings. So, to print the contents of the string register “*(BF” before the label, and the contents of the string register “*(AF” before the label, you would type:

<*(BF+*countername.symbolname*–*(AF>

A word of warning when quoting characters: *trnum* recognizes the quote character *only* when the next character begins a pair, or when within a pair. In these cases, it throws away the quote character and accepts the next character literally; any special effects

which would normally take place, such as beginning another part of the pair, are ignored. Further, if the quote character precedes a character which would normally begin a pair, the quote character is not printed, and no pair is begun. In all other cases, the quote character is simply copied through.

The characters which quote characters, begin and end pairs, and separate parts of a pair, may be reset; but they must not be reset to a character which is one of the control characters. Hence, you could reset the character indicating the beginning of a pair to "{", but not to "+". To get around this, you can "turn off" a control character by defining it to be null, that is,

```
.\" trnum: begin
```

**Acknowledgements**

This program grew out of a similar but simpler version written at Purdue University. Subhash Agrawal suggested the idea and helped debug and refine the original program.

**References**

[1] Ossanna, Joseph F., "NROFF/TROFF User's Manual," *internal memorandum*, Bell Laboratories, Murray Hill, NJ 07974 (January 1979).

[2] Tuthill, Bill, "Refer – A Bibliography System," *UNIX User's Manual, Volume 2*, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720 (July 1984).

[3] Kernighan, Brian W., "PIC – A Graphics Language for Typesetting User Manual," *internal memorandum*, Bell Laboratories, Murray Hill, NJ 07974 (March, 1982).

[4] Lesk, M. E., "Tbl – A Program to Format Tables," *internal memorandum*, Bell Laboratories, Murray Hill, NJ 07974 (January 1979).

[5] Kernighan, Brian W. and Cherry, Lorinda L., "Typesetting Mathematics – User's Guide (Second Edition)," *internal memorandum*, Bell Laboratories, Murray Hill, NJ

07974 (August 1978).

[6] Knuth, Donald E., *The TEXbook*, Addison-Wesley Publishing Company, (1984).

[7] Kernighan, Brian W., and Plauger, P. J., *The Elements of Programming Style*, McGraw-Hill, Inc., (1978).

[8] Rudin, Walter, *Principles of Mathematical Analysis*, Third Edition, McGraw-Hill, Inc., (1976).