© 1986 Gary Perlman

Section 5.1 Table of Analysis Programs

	Descriptive	Inferential	Graphical
Univariate			
`stats`	simple stats		
`desc`	many stats	t-test	histogram
`ts`	summary	auto-correlation	bar plot
Multilevel			
`oneway`	group stats	(un)weighted between anova	error barplots
`rankind`	rank stats	Mann-Whitney Kruskal-Wallis	fivenum plots
Bivariate			
`pair`	column stats differences correlation	paired t-test simple regression	scatter plot
Multivariate			
`regress`	variable stats correlation	linear regression partial correlation	residual output
`rankrel`	rank stats correlation	Wilcoxon Friedman	
Multifactor			
`anova`	cell stats	mixed model ANOVA	
`contab`	crosstabs	chi-square fisher exact test	

© 1986 Gary Perlman

Section 5.2 stats: print summary statistics

stats prints summary statistics for its input. Its input is a free format series of strings from which it extracts only numbers for analysis. When a full analysis is not needed, the following names request specific statistics.

```
n  min  max  sum  ss  mean  var  sd  skew  kurt  se  NA

prompt: stats
stats: reading input from terminal
1 2 3 4 5 6 7 8 9 10
EOF
n     =     10
NA    =     0
min   =     1
max   =     10
sum   =     55
ss    =     385
mean  =     5.5
var   =     9.16667
sd    =     3.02765
se    =     0.957427
skew  =     0
kurt  =     1.43836

prompt: series 1 100 | dm logx1 | stats min mean max
0     3.63739     4.60517
```

© 1986 Gary Perlman

Section 5.3 desc: descriptions of a single distribution

desc describes a single distribution. Summary statistics, modifiable format histograms and frequency tables, and single distribution *t*-tests are supported. **desc** reads free format input, with numbers separated by any amount of white space (blank spaces, tabs, newlines). When order statistics are being printed, or when a histogram or frequency table is being printed, there is a limit to the number of input values that must be stored. Although system dependent, usually several thousand values can be stored.

An example input to **desc** is shown below.

```
3 3 4 4 7 7 7 7 8 9 1 2 3 4 5 6 7
8 9 9 8 7 6 5 4 3 2 4 5 6 1 2 3 4 3  1  7 7
```

The call to **desc** includes many options: **-o** for Order statistics, **-hcfp** respectively for a Histogram, and a table with Cumulative Frequencies and Proportions, **-m 0.5** to set the Minimum allowed value to 0.5, **-M 8** to set the Maximum allowed value to 8, **-i 1** to set the Interval width in the histogram and table to 1, and **-t 5** to request a t-test with null mean equal to 5.

```
desc  -o  -hcfp  -m 0.5  -M 8  -i 1  -t 5
```

The output follows.

```
-------------------------------------------------------------
 Under Range    In Range  Over Range     Missing         Sum
           0          35           3           0     164.000
-------------------------------------------------------------
        Mean      Median    Midpoint   Geometric    Harmonic
       4.686       4.000       4.500       4.055       3.296
-------------------------------------------------------------
          SD   Quart Dev       Range     SE mean
       2.193       2.000       7.000       0.371
-------------------------------------------------------------
     Minimum  Quartile 1  Quartile 2  Quartile 3     Maximum
       1.000       3.000       4.000       7.000       8.000
-------------------------------------------------------------
        Skew     SD Skew    Kurtosis     SD Kurt
      -0.064       0.414       1.679       0.828
-------------------------------------------------------------
   Null Mean           t    prob (t)           F    prob (F)
       5.000      -0.848       0.402       0.719       0.402
-------------------------------------------------------------

Midpt    Freq     Cum    Prop     Cum
1.000       3       3   0.086   0.086 ***
2.000       3       6   0.086   0.171 ***
3.000       6      12   0.171   0.343 ******
4.000       6      18   0.171   0.514 ******
5.000       3      21   0.086   0.600 ***
6.000       3      24   0.086   0.686 ***
7.000       8      32   0.229   0.914 ********
8.000       3      35   0.086   1.000 ***
```

© 1986 Gary Perlman

Section 5.4 ts: time series analysis and plots

ts performs simple analyses and plots for time series data. Its input is a free format stream of at most 1000 numbers. It prints summary statistics for the time series, allows rescaling of the size of the time series so that time series of different lengths can be compared, and optionally computes auto-correlations of the series for different lags. Auto-correlation analysis detects recurring trends in data. For example, an auto-correlation of lag 1 of a time series pairs each value with the next in the series. **ts** is best demonstrated on an oscillating sequence, the output from which is shown below. The call to **ts** includes several options: **-c 5** requests autocorrelations for lags of 1 to 5, the **-ps** options request a time-series plot and statistics, and the **-w 40** option sets the width of the plot to 40 characters.

```
ts  -c 5  -ps  -w 40

1 2 3 4 5 4 3 2 1 2 3 4 5 4 3 2 1

n       = 17
sum     = 49
ss      = 169
min     = 1
max     = 5
range   = 4
midpt   = 3
mean    = 2.88235
sd      = 1.31731
Lag      r    r^2    n'            F    df      p
  0  0.000  0.000    17        0.000    15  1.000
  1  0.667  0.444    16       11.200    14  0.005
  2 -0.047  0.002    15        0.028    13  0.869
  3 -0.701  0.491    14       11.590    12  0.005
  4 -1.000  1.000    13        0.000    11  0.000
  5 -0.698  0.487    12        9.507    10  0.012
-----+------------|------------+--------
------------------|
         ---------|
                  |-
                  |-----------
                  |----------------------
                  |-----------
                  |-
         ---------|
------------------|
         ---------|
                  |-
                  |-----------
                  |----------------------
                  |-----------
                  |-
         ---------|
------------------|
-----+------------|------------+--------
1.000                          5.000
```

© 1986 Gary Perlman

Section 5.5 oneway: one way analysis of variance

oneway performs a between-groups one-way analysis of variance. It is partly redundant with **anova**, but is provided to simplify analysis of this common experimental design. The input to **oneway** consists of each group's data, in free input format, separated by a special value called the *splitter*. Group sizes can differ, and **oneway** uses a weighted or unweighted (Keppel, 1973) means solution. At most 20 groups can be compared. When two groups are being compared, the **-t** option prints the significance test as a *t*-test. The program is based on a method of analysis described by Guilford and Fruchter (1978).

An example interactive session with **oneway** is shown below. The call to **oneway** includes the **-s** option with 999 as the value of the Splitting value between groups. The **-u** option request the *unweighted* means solution rather than the default *weighted* means solution. The **-w 40** option requests an error bar plot of width 40. Meaningful names are given to the groups.

```
prompt: oneway  -s 999  -u  -w 40  less equal more
oneway: reading input from terminal:
1 2 3 4 5 4 3 2 1
999
3 4 5 4 3 4 5 4 3
999
7 6 5 7 6 5
EOF

Name          N     Mean       SD      Min      Max
less          9    2.778    1.394    1.000    5.000
equal         9    3.889    0.782    3.000    5.000
more          6    6.000    0.894    5.000    7.000
Total        24    4.000    1.642    1.000    7.000

less      |<-======(==#==)=======---->             |
equal     |             <===(=#)====->             |
more      |                          <===(==#=)===>|
           1.000                             7.000

Unweighted Means Analysis:
Source           SS    df         MS        F     p
Between      41.333     2     20.667   17.755 0.000 ***
Within       24.444    21      1.164
```

© 1986 Gary Perlman

Section 5.6 rankind: rank-order analysis of independent groups

rankind prints rank-order summary statistics and compares independent group data using non-parametric methods. It is the non-parametric counterpart to the normal theory **oneway** program, and the input format to **rankind** is the same as for **oneway**. Each group's data are in free input format, separated by a special value, called the *splitter*. Like **oneway**, there are plots of group data, but **rankind**'s show the minimum, 25th, 50th, and 75th percentiles, and the maximum. Significance tests include the median test, Fisher's exact test, Mann-Whitney U test for ranks, and the Kruskal-Wallis analysis of variance of ranks.

The following example is for the same data as in the example with **oneway**. The options to set the splitter and plot width are the same for both programs. Meaningful names are given to the groups.

```
prompt: rankind  -s 999  -w 40  less equal more
rankind: reading input from terminal:
1 2 3 4 5 4 3 2 1
999
3 4 5 4 3 4 5 4 3
999
7 6 5 7 6 5
EOF

             N   NA      Min      25%   Median      75%      Max
less         9    0     1.00     1.75     3.00     4.00     5.00
equal        9    0     3.00     3.00     4.00     4.25     5.00
more         6    0     5.00     5.00     6.00     7.00     7.00
Total       24    0     1.00     3.00     4.00     5.00     7.00

less      |<   ---------#------      >            |
equal     |            <-----#--    >            |
more      |                         <------#----->|
           1.000                              7.000

Median-Test:
                 less  equal   more
        above       1      2      6      9
        below       6      3      0      9
                    7      5      6     18
        WARNING: 6 of 6 cells had expected frequencies less than 5
        chisq       9.771429     df   2      p  0.007554

Kruskal-Wallis:
        H (not corrected for ties)             13.337778
        Tie correction factor                   0.965652
        H (corrected for ties)                 13.812197
        chisq      13.812197     df   2      p  0.001002
```

© 1986 Gary Perlman

Section 5.7 pair: paired points analysis and plots

pair analyzes paired data by printing summary statistics and significance tests for two input variables in columns and their difference, correlation and simple linear regression, and plots. The test of the difference of the two columns against zero is equivalent to a paired t-test. The input consists of a series of lines, each with two paired points. When data are being stored for a plot, at most 1000 points can be processed.

An example input to **pair** is generated using the **series** and **dm** programs connected by pipes. The input to **pair** are the numbers 1 to 100 in column 1, and the square roots of those numbers in column 2. The **dm** built-in variable **INLINE** is used in a condition to switch the sign of the second column for the second half of the data. **pair** reads X-Y points and predicts Y (in column 2) with X (in column 1). The significance test of the difference of the columns against 0.0 is equivalent to a paired-groups *t*-test. The call to **pair** includes several options: **-sp** requests Statistics and a Plot, **-w 40** sets the Width of the plot to 40 characters, and **-h 15** sets the Height of the plot to 15 characters.

```
series 1 100 | dm x1 "(INLINE>50?-1:1)*x1^.5" | pair -sp -w 40 -h 15

                Column 1   Column 2   Difference
Minimums          1.0000   -10.0000       0.0000
Maximums        100.0000     7.0711     110.0000
Sums           5050.0000  -193.3913    5243.3913
SumSquares   338350.0000  5049.9989  395407.6303
Means             50.5000    -1.9339      52.4339
SDs               29.0115     6.8726      34.8845
t(99)             17.4069    -2.8140      15.0307
p                  0.0000     0.0059       0.0000

Correlation    r-squared      t(98)            p
    -0.8226       0.6767   -14.3219       0.0000
  Intercept        Slope
     7.9070      -0.1949

|----------------------------------------|7.07107
|              323232                    |
|        123232                          |
|     2322                               |
|  223                                   |
|221                                     |
|1                                       |
|                                        |
|                                        |Column 2
|                                        |
|                                        |
|                                        |
|                                        |
|                    2322                |
|                       123232321        |
|                                223232323|
|----------------------------------------|-10
1.000                                100.000
           Column 1  r=-0.823
```

© 1986 Gary Perlman

Section 5.8 rankrel: rank-order analysis of related groups

rankrel prints rank-order summary statistics and compares data from related groups. It is the non-parametric counterpart to parts of the normal theory **pair** and **regress** programs. Each group's data are in a column, separated by whitespace. Instead of normal theory statistics like mean and standard deviation, the median and other quartiles are reported. Significance tests include the binomial sign test, the Wilcoxon signed-ranks test for matched pairs, and the Friedman two-way analysis of variance of ranks.

The following (transposed) data are contained in the file **siegel.79**, and are based on the example on page 79 of Siegel (1956). The astute analyst will notice that the last datum in column 2 in Siegel's book is misprinted as 82.

```
82  69    73    43    58    56    76    65
63  42    74    37    51    43    80    62
```

When the output contains a suggestion to consult a table of computed exact probability values, it is because the continuous chi-square or normal approximation may not be adequate. Siegel (1956) notes that the normal approximation for the probability of the computed Wilcoxon *T* statistic is *excellent* even for small samples such as the one above. Once again, the astute analyst will see the flaw in Siegel's analysis when he uses a normal approximation; he fails to use a correction for continuity.

```
prompt: rankrel control prisoner < siegel.79
             N   NA      Min      25%   Median      75%      Max
control      8    0    43.00    57.00    67.00    74.50    82.00
prisoner     8    0    37.00    42.50    56.50    68.50    80.00
Total       16    0    37.00    47.00    62.50    73.50    82.00

Binomial Sign Test:
        Number of cases control is above prisoner:   6
        Number of cases control is below prisoner:   2
        One-tail probability (exact)            0.144531

Wilcoxon Matched-Pairs Signed-Ranks Test:
    Comparison of control and prisoner
        T (smaller ranksum of like signs)       4.000000
        N (number of signed differences)        8.000000
        z                                       1.890378
        One-tail probability approximation      0.029354
        NOTE: Yates' correction for continuity applied
        Check a table for T with N = 8

Friedman Chi-Square Test for Ranks:
        Chi-square of ranks                     2.000000
        chisq       2.000000     df   1      p  0.157299
        Check a table for Friedman with N = 8

Spearman Rank Correlation (rho) [corrected for ties]:
        Critical r (.05) t approximation        0.706734
        Critical r (.01) t approximation        0.834342
        Check a table for Spearman rho with N = 8
        rho                                     0.785714
```

© 1986 Gary Perlman

Section 5.9 regress: multiple correlation/regression

regress performs a multiple linear correlation and regression analysis. Its input consists of a series of lines, each with an equal number of columns, one column per variable. In the regression analysis, the first column is predicted with all the others. There are options to print the matrix of sums of squares and the covariance matrix. There is also an option to perform a partial correlation analysis to see the contribution of individual variables to the whole regression equation. The program is based on a method of analysis described by Kerlinger & Pedhazur (1973). Non-linear regression models are possible using transformations with |STAT utilities like **dm**. The program can handle up to 20 input columns, but the width of the output for more than 10 is awkward.

The following artificial example predicts a straight line with a log function, a quadratic, and an inverse function. The input to **regress** is created with **series** and **dm**. The call to **regress** includes the **-p** option to request a partial correlation analysis and meaningful names for most of the variables in the analysis. The output from **regress** includes summary statistics for each variable, a correlation matrix, the regression equation and the significance test of the multiple correlation coefficient, and finally, a partial correlation analysis to examine the contribution of individual predictors, after the others are included in the model.

```
series 1 20 | dm x1 logx1 x1*x1 1/x1 | regress -p linear log quad inverse

Analysis for 20 cases of 4 variables:
Variable       linear         log        quad     inverse
Min             1.0000      0.0000      1.0000      0.0500
Max            20.0000      2.9957    400.0000      1.0000
Sum           210.0000     42.3356   2870.0000      3.5977
Mean           10.5000      2.1168    143.5000      0.1799
SD              5.9161      0.8127    127.9023      0.2235

Correlation Matrix:
linear          1.0000
log             0.9313      1.0000
quad            0.9713      0.8280      1.0000
inverse        -0.7076     -0.9061     -0.5639      1.0000
Variable       linear         log        quad     inverse

Regression Equation for linear:
linear  =  5.539 log  +  0.02245 quad  +  6.764 inverse  +  -5.66305

Significance test for prediction of linear
    Mult-R  R-Squared      SEest     F(3,16)   prob (F)
    0.9996     0.9993     0.1707   7603.7543     0.0000

Significance test(s) for predictor(s) of linear
Predictor      beta         b        Rsq         se      t(16)         p
log          0.7609    5.5389    0.9684    0.2709   20.4478    0.0000
quad         0.4854    0.0225    0.8795    0.0009   25.4555    0.0000
inverse      0.2555    6.7638    0.9314    0.6688   10.1139    0.0000
```

© 1986 Gary Perlman

Section 5.10 anova: multi-factor analysis of variance

anova performs analysis of variance with one random factor and up to nine independent factors. Both within-subjects and unequal-cells between-subjects factors are supported. Nested factors, other than those involving the random factor, are not supported. The input format is simple: each datum is preceded by a description of the conditions under which the datum was obtained. For example, if subject 3 took 325 msec to respond to a loud sound on the first trial, the input line to **anova** might be:

```
s3  loud  1  325
```

From input lines of this format, **anova** infers whether a factor is within- or between-subjects, prints cell means for all main effects and interactions, and prints standard format *F* tables with probability levels. The computations done in **anova** are based on a method of analysis described by Keppel (1973), however, for unequal cell sizes on between-groups factors, the weighted-means solution is used instead of Keppel's preferred unweighted solution. The weighted-means solution requires that sample sizes must be in constant proportions across rows and columns in interactions of between-subjects factors or else the analysis may be invalid.

An example input to **anova** is shown below. The call to **anova** gives meaningful names to the columns of its input. The output from **anova** contains cell statistics for all systematic sources (main effects and interactions), a summary of the design, and an F-table.

```
anova subject noise trial RT

s1   loud   1   259
s1   loud   2   228
s2   soft   1   526
s2   soft   2   480
s3   loud   1   325
s3   loud   2   315
s4   soft   1   418
s4   soft   2   397

SOURCE: grand mean
noise  trial     N       MEAN         SD         SE
                 8   368.5000   104.8713    37.0776

SOURCE: noise
noise  trial     N       MEAN         SD         SE
loud             4   281.7500    46.1257    23.0629
soft             4   455.2500    58.8749    29.4374

SOURCE: trial
noise  trial     N       MEAN         SD         SE
       1         4   382.0000   116.0603    58.0302
       2         4   355.0000   108.1943    54.0971

SOURCE: noise trial
noise  trial     N       MEAN         SD         SE
loud   1         2   292.0000    46.6690    33.0000
loud   2         2   271.5000    61.5183    43.5000
soft   1         2   472.0000    76.3675    54.0000
soft   2         2   438.5000    58.6899    41.5000
```

© 1986 Gary Perlman

```
FACTOR:    subject      noise      trial         RT
LEVELS:          4          2          2          8
TYPE  :     RANDOM    BETWEEN     WITHIN       DATA

SOURCE           SS  df            MS        F      p
=================================================
mean   1086338.0000   1  1086338.0000  145.111  0.007 **
s/n      14972.5000   2     7486.2500

noise    60204.5000   1    60204.5000    8.042  0.105
s/n      14972.5000   2     7486.2500

trial     1458.0000   1     1458.0000   10.942  0.081
ts/n       266.5000   2      133.2500

nt           84.5000   1       84.5000    0.634  0.509
ts/n       266.5000   2      133.2500
```

© 1986 Gary Perlman

Section 5.11 contab: contingency tables and chi-square

contab supports the analysis of multifactor designs with categorical data. Contingency tables (also called crosstabs) and chi-square test of independence are printed for all two-way interactions of factors. The method of analysis comes from several sources, especially Bradley (1968), Hays (1973), and Siegel (1956). The input format is similar to that of **anova**: each cell count is preceded by labels indicating the level at which that frequency count was obtained. Below are fictitious data of color preferences of boys and girls:

```
boys        red         3
boys        blue        17
boys        green       4
boys        yellow      2
boys        brown       10
girls       red         12
girls       blue        10
girls       green       5
girls       yellow      8
girls       brown       1
```

The output from the following command includes of a summary of the input design, tables, and statistical analyses.

```
contab  sex  color

FACTOR:        sex      color       DATA
LEVELS:          2          5         72

color      count
red           15
blue          27
green          9
yellow        10
brown         11
Total         72
        chisq      15.222222     df   4      p  0.004262

SOURCE: sex color
             red    blue   green  yellow   brown  Totals
boys           3      17       4       2      10      36
girls         12      10       5       8       1      36
Totals        15      27       9      10      11      72
Analysis for sex x color:
        WARNING: 2 of 10 cells had expected frequencies < 5
        chisq      18.289562     df   4      p  0.001083
        Cramer's V                             0.504006
        Contingency Coefficient                0.450073
```

© 1986 Gary Perlman

Section 5.12 dprime: d'/beta for signal detection data

dprime computes *d'* (a measure of discrimination of stimuli) and *beta* (a measure of response bias) using a method of analysis discussed in Coombs, Dawes, & Tversky (1970). The input to **dprime** can be a series of lines, each with two paired indicators: the first tells if a signal was present and the second tells if the observer detected a signal. From that, **dprime** computes the *hit-rate* (the proportion of times the observer detected a signal that was present), and the *false-alarm-rate* (the proportion of times the observer reported a signal that was not present). If the hit-rate and the false-alarm-rate are known, then they can be supplied directly to the program:

```
prompt: dprime .7 .4
  hr       far     dprime      beta
0.70      0.40       0.78      0.90
```

The input in raw form, with the true stimulus (Was a signal present or just noise?) in column 1 and the observer's response (Did the observer say there was a signal?) in column 2, is followed by the output.

```
signal      yes
signal      yes
signal      yes
signal      yes
signal      yes
signal      yes
signal      yes
signal      no
signal      no
signal      no
noise       yes
noise       yes
noise       no
noise       no
noise       no
```

dprime would produce for the above data:

```
           signal   noise
yes          7        2
 no          3        3

  hr       far     dprime      beta
0.70      0.40       0.78      0.90
```

© 1986 Gary Perlman

Section 5.13 CALC: Tutorial and Manual

calc is a program for mathematical calculations for which you might use a hand-held calculator. **calc** supplies most of the operations common to programming languages and variables with constraint properties much like those in spreadsheets.

The arithmetical operators **calc** offers are

+	addition
-	subtraction and change-sign
*****	multiplication
/	division
%	modulo division
^	exponentiation

Arithmetical expressions can be arbitrarily complex and are generally evaluated left to right. That is,

```
a + b - c
```

is the same as

```
(a + b) - c
```

Exponentiation is evaluated before multiplication and division which are evaluated before addition and subtraction. For example, the expression

```
a + b - c * d / e ^ 2
```

is parsed as

```
(a + b) - ((c * d) / (e ^ 2))
```

This default order of operations can be overridden by using parentheses.

calc supplies some transcendental functions: **sqrt**, **log**, **exp**, and **abs**, and the following trigonometric functions: **sin**, **asin**, **cos**, **acos**, **tan**, and **atan**, for which degrees are measured in radians.

Using CALC

To use **calc**, begin by typing

```
calc
```

at the command level, and **calc** will prompt you with

```
CALC:
```

You can supply inputs to **calc** from files specified by command line arguments. For example, typing

```
calc foo
```

will read from the file **foo** and then ask for input from you. Type in each of your expressions followed by **RETURN** and **calc** will respond with how it parsed your expression followed by the result. In all following examples, what you would type in is preceded by the **calc** prompt

```
CALC:
```

and what **calc** responds with is immediately after. A simple calculation is:

```
CALC: sqrt (12^2 + 5^2)
sqrt(((12 ^ 2) + (5 ^ 2)))     = 13
```

© 1986 Gary Perlman

Expressions can be stored by assigning them to variables. For example you could type:

```
CALC: pi = 22/7
(22 / 7)      = 3.14286
CALC: pi
pi          = 3.14286
```

Variables can be used in expressions.

```
CALC: area = pi * r^2
(pi * (r ^ 2))      = UNDEFINED
CALC: area
area     = UNDEFINED
```

area is undefined because **r** has not been set. Once **r** is set, **area** will have a value because **area** is set to an equation rather than a particular value. This can be observed by printing all the variables so far introduced with **^V**, which may have to be typed twice as **^V** is used in some UNIX versions to quote characters.

```
CALC: ^V
pi       =      3.14286 = (22 / 7)
area     =    UNDEFINED = (pi * (r ^ 2))
r        =    UNDEFINED =
```

The variable table is formatted so that each variable's name is on the left, followed by its current value, followed by its current definition. If **r** is set to 5, the value of **area** is now defined.

```
CALC: r = 5
5        = 5
CALC: ^V
pi       =      3.14286 = (22 / 7)
area     =      78.5714 = (pi * (r ^ 2))
r        =            5 = 5
```

The effect of changing **r** on **area** can be observed because of the way **area** is defined.

```
CALC: r = 2
2         = 2
CALC: area
area      = 12.5714
```

A special variable named **$** is always equal to the most recent result printed.

Setting Constant Values

Of course, there are times when you want to set a variable to a value and not have it depend on the values of variables at a later time. To do this, you precede an expression with the number operator **#**. For example,

```
CALC: area2 = # area
12.5716      = 12.5716
CALC: ^V
pi       =      3.14286 = (22 / 7)
area     =      12.5716 = (pi * (r ^ 2))
r        =            2 = 2
area2    =      12.5716 = 12.5716
```

area2 does not depend on the variable to which it was set because the number operator **#** only lets numbers through it rather than expressions. If **area2** was set without the **#** operator, it would be subject to any changes in **area** or to any changes in variables on which **area** depends.

```
CALC: area2 = area
area      = 12.5716
CALC: ^V
```

© 1986 Gary Perlman

```
pi       =      3.14286 = (22 / 7)
area     =      12.5716 = (pi * (r ^ 2))
r        =            2 = 2
area2    =      12.5716 = area
```

Testing Conditions

Variables can be set based on a tested condition. For example, you may want a variable **max** to always be the maximum of **a** and **b**.

```
CALC: max = if a > b then a else b
(if (a > b) then a else b)    = UNDEFINED
```

max is undefined because **a** and **b** have not been set.

```
CALC: a = 21
21     = 21
CALC: b = 3^3
(3 ^ 3)    = 27
CALC: max
max      = 27
CALC: a = 50
50   = 50
CALC: max
max      = 50
```

The if-then-else expression allows variables to be set based on conditions. This condition can be made up with relational and logical operators. The relational operators available with **calc** are:

==	test equality
!=	test inequality
>=	greater than or equal
<=	less than or equal
>	greater than
<	less than

while the logical operators are:

&	and
|	or
!	not

A more complicated expression involving these is:

```
if a > b & b > c then b
```

The **else** part of the conditional is optional, and if not present and the condition is false, the conditional is undefined.

Undefined Variables

Variables are undefined if they have not been set, if they depend on variables that are undefined, or if they are set to an expression involving an illegal operation.

```
CALC: 1/0
(1 / 0)     = UNDEFINED
```

You can be confident that no operations will result in **calc** blowing up. Thus you could write the equation for the roots of a quadratic formula with the following definitions and always get reasonable answers.

```
x = 0
a = b = 1
```

© 1986 Gary Perlman

```
c = -1
radical = sqrt (b^2 - 4*a*c)
equation = a*x^2 + b*x + c
derivative = 2*a*x + b
root1 = (-b + radical) / (2 * a)
root2 = (-b - radical) / (2 * a)
```

Control Characters

Non-mathematical operations are accomplished with control characters. To type a control character, say CTRL-p, while you hold down the key labeled CTRL you type a **p**. This will appear as **^P**. Some control characters have special meanings, such as "stop the program" so you must be careful with them. On UNIX, you can avoid some problems with control characters by typing a **^V** before them. This character removes any special meaning associated with the character immediately following it. So to type **^P** you could be extra safe and type **^V^P**. To type a **^V**, you may have to type it twice. Unfortunately, these conventions are not universal.

The following control operations are available with **calc**.

^P	toggle the printing of expressions (UNIX only)
^Rf	read the input from file f and return to current state
^V	print the variable table
^Wf	write the variable table to file **f**
	(**^W** is a synonym for **^V**)

If you forget any of these commands, you can type a **?** to get **calc** to remind you.

© 1986 Gary Perlman

Table of `calc` Operations

Operator	Precedence	Associativity	Description
`$`	const	none	numerical value of previous calculation
`#a`	1	none	numerical value of a
`a=b`	2	right	a is set to expression b
`if a then b`	3	left	if a != 0 then b else `UNDEFINED`
`else`	4	left	
`a|b`	5	left	true if a or b is true
`a&b`	6	left	true is a and b are true
`!a`	7	none	true is a is false
`a==b`	8	none	true if a equals b
`a!=b`	8	none	true if a is not equal b
`a<b`	8	none	true if a is less than b
`a>b`	8	none	true if a greater than b
`a>=b`	8	none	true if a > b | a == b
`a<=b`	8	none	true if a < b | a == b
`a+b`	9	left	a plus b
`a-b`	9	left	a minus b
`a*b`	10	left	a times b
`a/b`	10	left	a divided by b
`a%b`	10	left	a modulo b
`a^b`	11	right	a to the b
`-a`	12	none	change sign
`abs(a)`	12	none	absolute value
`exp(a)`	12	none	e to the power a
`log(a)`	12	none	natural logarithm of a
`sqrt(a)`	12	none	square root of a
`sin(a)`	12	none	sine of a in radians (`cos` & `tan`)
`asin(a)`	12	none	arc sine of a (`acos` & `atan`)

© 1986 Gary Perlman

CHAPTER 5

Data Analysis

5.1 **Table of Analysis Programs** 5-1
5.2 **stats: print summary statistics** 5-2
5.3 **desc: descriptions of a single distribution** 5-3
5.4 **ts: time series analysis and plots** 5-4
5.5 **oneway: one way analysis of variance** 5-5
5.6 **rankind: rank-order analysis of independent groups** 5-6
5.7 **pair: paired points analysis and plots** 5-7
5.8 **rankrel: rank-order analysis of related groups** 5-8
5.9 **regress: multiple correlation/regression** 5-9
5.10 **anova: multi-factor analysis of variance** 5-10
5.11 **contab: contingency tables and chi-square** 5-12
5.12 **dprime: d'/beta for signal detection data** 5-13
5.13 **CALC: Tutorial and Manual** 5-14
Using CALC
Setting Constant Values
Testing Conditions
Undefined Variables
Control Characters

Each of the analysis programs are introduced, showing some, but not all of their options. Full documentation can be found in the manual entries. Details about the procedures and assumptions are found in the references in the ALGORITHM sections of the manual entries. Most analysis programs allow summary statistics, inferential statistics. and simple graphics. In general, a program consists of all the analyses for a specific type of data. There are programs for univariate (single) distributions, multilevel, and multifactor analysis. Some simple analyses are possible by combining data manipulation and analysis programs. For example, Scheffe confidence intervals can be computed for means using the **`probdist`** and **`calc`** programs. A tutorial and reference manual for **`calc`** is the final section of this chapter.

© 1986 Gary Perlman