TRUE BASIC

The Original BASIC

Business & Scientific Graphics Toolkit

Reference Guide

Graphics Libraries

This section describes subroutines for generating complicated graphical displays of data. The subroutines are contained in three library files:

BGLIB.TRC	for drawing pie charts, bar charts, and histograms; including the routines BARCHART, HISTOGRAM, IBEAM, MULTIBAR, MULTIHIST, PIECHART, and several ASK... and SET... routines for finding out about or setting attributes of graphs.
SGLIB.TRC	for plotting data and function values; including the routines ADDDATAGRAPH, ADDFGRAPH, ADDLSGRAPH, DATAGRAPH, FGRAPH, MANYDATAGRAPH, MANYFGRAPH, SORTSPOINTS, and many ASK... and SET... routines for finding out about or setting attributes of graphs
SGFUNC.TRC	for plotting values of functions that you define: ADDFGRAPH, FGRAPH, MANYFGRAPH

The graphics subroutines are described below, in alphabetical order.

ADDDATAGRAPH Subroutine

Library: SGLIB.TRC

Syntax: CALL ADDDATAGRAPH (*numarrarg*,, *numarrarg*, *numex*, *numex*, *strex*)

numarrarg:: *numarr*
numarr bowlegs

Usage:
```
CALL ADDDATAGRAPH (x(), y(), pstyle, lstyle, colors$)
```

Summary: Draws another line graph of a set of data points over the current graph.

Details: The **ADDDATAGRAPH** subroutine draws a line graph of the set of data points whose coordinates are represented by the values of the `x` and `y` arrays over the current graph; it simply adds the new graph to the existing graph. Therefore, portions of the added data graph may lie off the graph.

The `x` array contains the points' x-coordinates, and the `y` array contains their y-coordinates. The coordinates in the two arrays are matched according to their subscripts; that is, the elements with subscripts of 1 within both arrays are interpreted as the coordinates of a single point, as are the elements with subscripts of 2, and so on. Thus, the `x` and `y` arrays must have the same upper and lower bounds, or an error will be generated.

The value of `pstyle` determines the point style that will be used to draw the data points which comprise the graph. The allowable values for `pstyle` are summarized in the following table:

Available Point Styles

Value of `pstyle`	Resulting Point
0	No point (display line only)
1	Dot
2	Plus sign
3	Asterisk
4	Circle
5	X
6	Box
7	Up triangle
8	Down triangle
9	Diamond
10	Solid Box
11	Solid up triangle
12	Solid down triangle
13	Solid diamond

The value of `lstyle` determines the line style that will be used to connect the data points which comprise the graph. The allowable values for `lstyle` are summarized in the following table:

Available Line Styles

Value of `lstyle`	Resulting Line
0	No line (display points only)
1	Solid line
2	Dashed line
3	Dotted line
4	Dash-dotted line

The graph is actually composed of a series of line segments connecting the data points. You can suppress the display of the data points by passing a value of 0 in `pstyle`, or you can suppress the display of the connecting line segments by passing a value of 0 in `lstyle`.

Note that the **ADDDATAGRAPH** subroutine draws and connects the points in the order in which they are stored in the `x` and `y` arrays. If your points are not stored in left to right order, you may wish to use the **SORTPOINTS** subroutine to order the points before passing them to the **ADDDATAGRAPH** subroutine.

The value of `colors$` determines the color that will be used to draw the new graph. It generally consists of a single color name (in any combination of uppercase or lowercase letters). The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

Note: the color "BACKGROUND" refers to the current background color.

The value of `colors$` may also contain a color number instead of a color name, allowing you to access any of the colors supported by the current computer system.

Note that the **ADDDATAGRAPH** subroutine assumes that a graph currently exists which has been created by an invocation of the **FGRAPH** or **DATAGRAPH** subroutine. The **ADDDATAGRAPH** subroutine simply adds the line representing the specified data points to the existing graph; it does not rescale the graph or redraw the labels or title. If you attempt to invoke the **ADDDATAGRAPH** subroutine when a suitable graph has not already been displayed, an error will be generated.

Example: The following program, SGData2.TRU, can be found in the directory TBDEMOS:

```
!  SGData2   Chris' & Dave's cars' mileage.

!  Both drove the same kind of car.  Notice that one car's mileage
!  goes up and down depending on the season (winter is low).
!  The other is less affected.  Also, notice a few erroneous
!  data points!

LIBRARY "..\TBLibs\SGLib.trc"

DIM cmiles(0 to 200), cgallons(200), cmpg(200)
DIM dmiles(0 to 200), dgallons(200), dmpg(200)

CALL ReadData (cmiles, cgallons, cmpg)
CALL ReadData (dmiles, dgallons, dmpg)

CALL SetText ("Gas Mileage", "Miles Driven (Thousands)", "MPG")
CALL DataGraph (cmiles, cmpg, 0, 3, "red green yellow")
CALL AddDataGraph (dmiles, dmpg, 0, 1, "green")
```

```
GET KEY key

SUB ReadData (miles(), gallons(), mpg())

    READ miles(0)
    LET n = 0
    DO
       LET n = n + 1
       READ miles(n), gallons(n)
    LOOP until miles(n) = 0
    LET n = n - 1

    FOR i = 1 to n
        LET mpg(i) = (miles(i) - miles(i-1)) / gallons(i)
    NEXT i
    MAT redim mpg(n), miles(1:n)
    MAT miles = (1/1000) * miles

END SUB

! Chris's car

DATA 677.3                          ! first recorded

DATA 1104.9,9.5,  1567.6,9.3,  1869.7,6.7,  2179.5,6.0
DATA 2564.2,8.0,  2812.3,4.7,  3192.0,7.8,  3540.4,7.4
DATA 4044.4,10.2, 4317.5,5.8,  4747.8,8.7,  4946.2,3.7
DATA 5406.7,9.6,  5870.0,10.1, 6344.2,10.0, 6789.3,9.6
DATA 7208.1,9.1,  7624.8,9.6,  7786.6,3.2,  8244.3,9.4
DATA 8614.1,8.6,  9050.0,9.5,  9584,13.2,   9991.6,9.3
DATA 10389,9.4,   10804.4,9.1, 11216.1,10.3,11623.4,10.1
DATA 11970.4,9.54,12215.5,6.6, 12599.8,9.6, 12921.9,8.84
DATA 13238.1,7.7
DATA 13815.0,14.3,14170.0,8.8, 14531.0,8.3, 14880.9,9.0
DATA 15671,8.95, 16065,8.2,    16453,8.47,  16696,5.4
DATA 17144,8.8,  17568,9.1,    17997,8.65,  18450,9.3
DATA 18934,9.9,  19356,8.7,    19787,8.4,   20162,7.4
DATA 20572,8.25, 21025,8.8,    21345,9.0    ! did I read this right?
DATA 21713,5.0,  22043,6.6,    22514,9.2,   22968,9.6
DATA 23450,9.1,  23923,9.5,    24302,7.2,   24814,9.9
DATA 25272,9.1,  25738,9.0,    26128,7.7,   26603,8.9
DATA 26975,7.45, 27145,3.772
DATA 27523,7.36, 27834,6.4,    28266,8.4,   28652,8.3
DATA 29091,8.7,  29510,8.8,    29818,6.4,   30223,8.48
DATA 30626,8.9,  31056,8.24       ! ?
DATA 31410,8.16, 31786,8.6,    32161,8.9    ! ?
DATA 32550,9.2,  32941,9.045,  33302,9.3
DATA 0,0

! Dave's car

DATA 0                              ! full tank on delivery
DATA 272,6.35,   599,6.56,     924,7.44,   1281,7.56
DATA 1462,4.47,  1705,4.32,    2099,8.02,  2673,12.03
DATA 3090,8.76,  3537,8.6,     3991,9.28,  4419,8.73
DATA 4779,7.86,  5022,5.4,     5407,7.88,  5731,7.3
DATA 6049,7.04,  6388,7.61,    6836,8.56,  7204,7.87
DATA 7633,9.21,  8000,7.93,    8455,9.52,  8765,7.17
DATA 9188,9.2,   9578,9.21,    10111,13.7, 10551,10.13
DATA 10884,6.16, 11261,8.16,   11550,7.01, 11888,8.43
DATA 12255,6.79, 12690,8.11,   13237,10.8, 13563,6.47
DATA 14036,8.89, 14418,8.91,   14758,7.28, 15183,9.16
DATA 15757,11,   16394,12.75,  16752,7.95, 17108,6.83
```

```
DATA 17543,9.01, 17943,9.48,   18362,8.88, 18781,9.07
DATA 19179,8.83, 19361,4.63,   19600,6.07, 19898,6.57
DATA 0,0

END
```

produces a graph comparing the fuel economy of two cars.

Exceptions:

100	Graph's title is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
110	Data arrays have different bounds in DataGraph
117	Can't handle this graph range: *low* to *high*.
11008	No such color: *color*.

See also: **DATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**

ADDFGRAPH Subroutine

Library: SGFUNC.TRC, SGLIB.TRC

Syntax: CALL ADDFGRAPH (*numex*, *strex*)

Usage:

```
CALL ADDFGRAPH (style, color$)
```

Summary: Draws another line graph of an externally defined function over the current graph.

Details: The **ADDFGRAPH** subroutine draws a line graph of the function F(x) over the current graph. The **ADDFGRAPH** subroutine does not change the scale of the current graph; it simply adds the new graph to the existing graph. Therefore, parts of the new function may be off the graph.

The function F(x) must be defined external to your main program. That is, it must be defined using a **DEF** statement or a **DEF** structure which appears after the **END** statement. The function you define must be defined over the entire domain specified. If it is not, the **ADDFGRAPH** subroutine may generate an error or draw the graph incorrectly.

Note that both the **ADDFGRAPH** subroutine and the **FGRAPH** subroutine utilize an externally defined function named F. Since a program may not contain two defined functions with the same name, it is your responsibility to ensure that the function F(x) is defined to calculate two different functions if you plan to use the **ADDFGRAPH** subroutine after calling the **FGRAPH** subroutine. (See the following example for one method of accomplishing this.)

The value of `style` determines the line style that will be used to connect the data points which comprise the graph. The allowable values for `style` are summarized in the following table:

Available Line Styles

Value of `style`	Resulting Line
0	No line (display points only)
1	Solid line
2	Dashed line
3	Dotted line
4	Dash-dotted line

The graph is actually composed of a series of short line segments. You can control the number of line segments used to display a graph with the **SETGRAIN** subroutine. Using more line segments creates a smoother graph, but takes longer to draw.

The value of `color$` determines the color that will be used to draw the new graph. It generally consists of a single color name (in any combination of uppercase or lowercase letters). The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `color$` may also contain a color number instead of a color name, allowing you to access any color supported by the current computer system.

If the value of `color$` contains more than one color, only the last color in the list will be used.

Note that the **ADDFGRAPH** subroutine assumes that a graph currently exists which has been created by an invocation of the **FGRAPH** or **DATAGRAPH** subroutine. The **ADDFGRAPH** subroutine simply adds the line representing the current function F(x) to the existing graph; it does not rescale the graph or redraw the labels or title. If you attempt to invoke the **ADDFGRAPH** subroutine when a suitable graph has not already been displayed, an error will be generated.

Example: The following program, SGFunc2.TRU, can be found in the directory TBDEMOS:

```
!  SGFunc2  Graph sine and cosine functions.

LIBRARY "..\TBLibs\SGFunc.trc", "..\TBLibs\SGLib.trc"

PUBLIC flag

CALL SetText ("Sine and Cosine Waves", "X Values", "Y Values")

CALL Fgraph (-2*pi, 2*pi, 1, "white white magenta")

LET flag = 1
CALL AddFgraph (2, "cyan")

GET KEY key

END

DEF F(x)
    DECLARE PUBLIC flag
    IF flag = 0 then LET F = Sin(x) else LET F = Cos(x)
END DEF
```

produces a graph of the functions Sin(x) and Cos(x). Notice the use of the public variable `flag` to change the behavior of the defined function being graphed.

Exceptions:
118 No canvas window yet.
11008 No such color: *color*.

See also: **SETGRAIN**, **FGRAPH**, **MANYFGRAPH**

ADDLSGRAPH Subroutine

Library: SGLIB.TRC

Syntax: CALL ADDLSGRAPH (*numarrarg*, *numarrarg*, *numex*, *strex*)

numarrarg:: *numarr*
numarr bowlegs

Usage: `CALL ADDLSGRAPH (x(), y(), style, color$)`

Summary: Computes and draws the least-squares linear fit for the specified points.

Details: The **ADDLSGRAPH** subroutine calculates and draws the least-squares linear fit of a set of data points.

The least-squares linear fit of a set of data points is the straight line which best fits the locations of those data points. That is, the least-squares linear fit of a set of data points is the straight line which minimizes the vertical distance between itself and each of the data points. Such a line may be used to help predict where data points might lie in areas for which data is unavailable.

The set of data points is specified as pairs of coordinates passed as the contents of the `x` and `y` arrays. The `x` array contains the points' x-coordinates, and the `y` array contains their y-coordinates. The coordinates in the two arrays are matched according to their subscripts; that is, the elements with subscripts of 1 within both arrays are interpreted as the coordinates of a single point, as are the elements with subscripts of 2, and so on. Thus, the `x` and `y` arrays must have the same upper and lower bounds, or an error will be generated.

The value of `style` determines the line style that will be used to draw the linear fit. The allowable values for `style` are summarized in the following table:

Available Line Styles

Value of `lstyle`	**Resulting Line**
0	No line (display points only)
1	Solid line
2	Dashed line
3	Dotted line
4	Dash-dotted line

The value of `color$` determines the color that will be used to draw the linear fit. It generally consists of a single color name (in any combination of uppercase or lowercase letters). The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

Note: the color "BACKGROUND" refers to the current background color.

The value of `color$` may also contain a color number instead of a color name, allowing you to access any of the colors supported by the current computer system.

Note that the **ADDLSGRAPH** subroutine assumes that a graph currently exists which has been created by an invocation of one of the various graphing subroutines. The **ADDLSGRAPH** subroutine simply adds the line representing the specified data points to the existing graph; it does not rescale the graph or redraw the labels or title. If you attempt to invoke the **ADDLSGRAPH** subroutine when a suitable graph has not already been displayed, an error will be generated.

Example: The following program, SGLSquar.TRU, can be found in the directory TBDEMOS:

```
!  SGLSquar   Add a least-squares line to data points.

!  Data taken from "The Shortwave Propagation Handbook" (2nd ed)
!  by George Jacobs and Theodore J. Cohen.  Page 111.

LIBRARY "..\TBLibs\SGLib.trc"

DIM x(120), y(120)

MAT READ x, y                    ! Data later

CALL SetYscale (70, 170)

CALL SetText ("Sunspots vs. Solar Flux", "Daily Sunspot Number",
"Daily Solar Flux")
CALL DataGraph (x, y, 6, 0, "red green yellow")
CALL AddLSgraph (x, y, 1, "red")
```

```
DATA  16,  17,   5,   4,  18,  19,  21,  24,  22,  25
DATA  28,  30,  32,  33,  31,  35,  21,  25,  26,  30
DATA  28,  31,  37,  37,  39,  38,  34,  25,  40,  41
DATA  43,  44,  42,  45,  47,  48,  50,  50,  52,  56
DATA  57,  59,  46,  42,  41,  45,  48,  52,  44,  45
DATA  49,  55,  58,  59,  53,  55,  55,  59,  57,  65
DATA  64,  61,  63,  64,  66,  65,  67,  69,  71,  76
DATA  75,  81,  80,  80,  81,  82,  87,  90,  84,  84
DATA  64,  65,  78,  78,  73,  80,  77,  74,  70,  70
DATA  61,  63,  73,  74,  73,  77,  79,  78,  79,  63
DATA  81,  94,  97,  93,  93,  86,  79,  98,  93, 116
DATA 116, 115, 116, 104, 127, 125, 130, 131, 123, 139

DATA  81,  84,  84,  88,  89,  87,  90,  89,  87,  87
DATA  85,  82,  91,  90,  87,  85,  96,  95,  95,  99
DATA  93,  94,  95,  98,  96, 103, 105, 111, 100,  94
DATA  99,  97,  97,  94,  97,  98, 100,  95,  97, 102
DATA 104, 104, 104, 105, 107, 109, 108, 108, 112, 115
DATA 115, 115, 116, 117, 120, 119, 127, 125, 133, 103
DATA 106, 110, 108, 111, 108, 107, 108, 107, 108, 105
DATA 110, 102, 107, 108, 108, 106, 110, 114, 118, 119
DATA 116, 115, 119, 118, 116, 114, 115, 114, 121, 122
DATA 126, 127, 125, 128, 131, 126, 127, 131, 130, 133
DATA 131, 129, 131, 123, 135, 138, 140, 144, 146, 148
DATA 158, 157, 156, 157, 154, 159, 159, 163, 162, 166

GET KEY key

END
```

produces a graph with a least-squares linear fit superimposed over it.

Exceptions: None

See also: **SETLS**, **ASKLS**, **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**

ASKANGLE Subroutine

Library: SGLIB.TRC

Syntax: CALL ASKANGLE (*strex*)

Usage: `CALL ASKANGLE (measure$)`

Summary: Reports the manner in which subsequent polar graphs drawn by the various data and function plotting subroutines will interpret angle measurements.

Details: The **ASKANGLE** subroutine is used to report the manner in which subsequent data and function polar plots produced by the **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**, **ADDFGRAPH**, and **MANYFGRAPH** subroutines will interpret angle measurements.

If the value of `measure$` is returned equal to "DEG" these subroutines will interpret angular coordinates for polar graphs as degrees. If the value of `measure$` is returned equal to "RAD" these subroutines will interpret angular coordinates for polar graphs as radians.

Note that the **ASKANGLE** subroutine only reports the interpretation of angular coordinates by polar graphs. Use the **ASKGRAPHTYPE** subroutine to report whether or not subsequent graphs will be drawn as polar graphs.

You can use the **SETANGLE** subroutine to control the manner in which the next data or function polar plot will interpret angular coordinates.

Example: None

Exceptions: None

See also: **SETANGLE**, **SETGRAPHTYPE**, **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**, **ADDFGRAPH**, **MANYFGRAPH**

ASKBARTYPE Subroutine

Library: BGLIB.TRC

Syntax: CALL ASKBARTYPE (*strvar*)

Usage: `CALL ASKBARTYPE (type$)`

Summary: Reports the arrangement of the bars within each group of subsequently drawn multiple bar chart or histogram.

Details: The **ASKBARTYPE** subroutine is used to report the arrangement of the bars within each group of a bar chart or histogram that will produced by a subsequent invocation of the **MULTIBAR** or **MULTIHIST** subroutine.

Both the **MULTIBAR** and **MULTIHIST** subroutines draw multiple bar-based graphs in a single frame. In such a graph, bars associated with a particular unit are grouped together.

The **ASKBARTYPE** subroutine allows you to report how the bars in each group will be arranged by returning one of the following values in `type$`:

Types of Bar Groupings

`Type$` value	Description
`"SIDE"`	Bars arranged side by side with space between them
`"STACK"`	Bars stacked one above the other
`"OVER"`	Bars arranged side by side but overlapped slightly

By default, the bar type is set to a value of `"SIDE"`. You can use the **SETBARTYPE** subroutine to change the current bar type setting.

Example: None

Exceptions: None

See also: **SETBARTYPE**, **MULTIBAR**, **MULTIHIST**

ASKGRAIN Subroutine

Library: SGLIB.TRC

Syntax: CALL ASKGRAIN (*numvar*)

Usage: `CALL ASKGRAIN (grain)`

Summary: Reports the grain with which subsequent invocations of the various function plotting subroutines will draw the line graph.

Details: The **ASKGRAIN** subroutine reports the `grain` with which subsequent invocations of the **FGRAPH**, **ADDFGRAPH**, and **MANYFGRAPH** subroutines will draw the line representing the function.

These subroutines actually graph the curve of the function which they are plotting as a series of line segments. The ***grain*** controls the number of line segments used to form each graphed curve. The higher the value of the grain, the more line segments are used and the smoother the resulting curve appears. However, higher grains also mean more work for the computer, and this means that each curve takes longer to draw.

By default, the **FGRAPH**, **ADDFGRAPH**, and **MANYFGRAPH** subroutines use a grain value of 64, which means that each line graph is composed of 64 individual line segments. This value strikes a generally acceptable balance of smoothness and speed, but this value can be changed using the **SETGRAIN** subroutine.

Example: None

Exceptions: None

See also: **SETGRAIN**, **FGRAPH**, **ADDFGRAPH**, **MANYFGRAPH**

ASKGRAPHTYPE Subroutine

Library: SGLIB.TRC

Syntax: CALL ASKGRAPHTYPE (*strvar*)

Usage: `CALL ASKGRAPHTYPE (type$)`

Summary: Reports the type of graph that will be drawn by subsequent data and function plotting subroutines.

Details: The **ASKGRAPHTYPE** subroutine is used to report the type of graph that will be produced for subsequent data and function plots produced by the **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**, **ADDFGRAPH**, and **MANYFGRAPH** subroutines.

The type of subsequent graphs is reported as the value of `type$`. The possible values of `type$` are:

Types of Graphs

`Type$` **value**	**Description**
`"XY"`	Normal graph
`"LOGX"`	Semi-logarithmic graph with x-axis logarithmically scaled
`"LOGY"`	Semi-logarithmic graph with y-axis logarithmically scaled
`"LOGXY"`	Logarithmic graph with both x- and y-axes logarithmically scaled
`"POLAR"`	Polar graph

You can use the **SETGRAPHTYPE** subroutine to control the type of graph that will be used for the next data or function plot.

Example: None

Exceptions: None

See also: **SETGRAPHTYPE**, **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**, **ADDFGRAPH**, **MANYFGRAPH**

ASKGRID Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL ASKGRID (*strvar*)

Usage: `CALL ASKGRID (style$)`

Summary: Reports the presence, direction, and type of the grid that will be used in subsequently drawn charts and graphs.

Details: The **ASKGRID** subroutine is used to report on the presence, direction, and type of the grid that will be drawn within the frame of graphs or charts produced by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH** subroutines.

The **ASKGRID** subroutine reports the presence and direction of the grid lines by returning one of the following values in `style$`:

Available Grid Directions

`Style$` **value**	**Description**
`""`	No grid lines
`"H"`	Horizontal grid lines only
`"V"`	Vertical grid lines only
`"HV"`	Both horizontal and vertical grid lines

The returned value of `style$` may also include instructions that indicate the type of grid lines that will be drawn. These instructions take the form of special characters appended to the letter (or letters) in the returned value of `style$`. If no such modifiers are present, grid lines will be drawn as solid lines. The following modifiers are possible:

Available Grid Type Modifiers

Modifier	Description
-	Dashed grid lines
.	Dotted grid lines
-.	Dash-dotted grid lines

For example, a value of `"H-.V"` would indicate that dash-dotted grid lines will be used in the horizontal direction and solid grid lines will be used in the vertical direction.

By default, the grid lines are turned off. You can use the **SETGRID** subroutine to change the current grid setting.

Example: None

Exceptions: None

See also: **SETGRID**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

ASKHLABEL Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL ASKHLABEL (*strvar*)

Usage:
```
CALL ASKHLABEL (hlabel$)
```

Summary: Reports the value of the horizontal label which will be displayed for subsequently drawn charts and graphs.

Details: The **ASKHLABEL** subroutine is used to report the value of the horizontal label that will be used to label the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, and **MANYDATAGRAPH** subroutines.

The **ASKHLABEL** subroutine returns the value of the horizontal label as `hlabel$`.

You may report the current values for the title, the horizontal label, and the vertical label simultaneously using the **ASKTEXT** subroutine. Use the **ASKVLABEL** and **ASKTITLE** subroutines to report the values of the vertical label and the title, respectively.

You may use the **SETHLABEL** subroutine to set the current value of the horizontal label.

Example: None

Exceptions: None

See also: **SETHLABEL**, **ASKTEXT**, **ASKVLABEL**, **ASKTITLE**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **PIECHART**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

ASKLAYOUT Subroutine

Library: BGLIB.TRC

Syntax: CALL ASKLAYOUT (*strvar*)

Usage:
```
CALL ASKLAYOUT (direction$)
```

Summary: Reports the direction of the bars within subsequently drawn bar charts and histograms.

Details: The **ASKLAYOUT** subroutine is used to report the direction of the bars within each bar chart or histogram produced by a subsequent invocation of the **MULTIBAR** or **MULTIHIST** subroutine.

The **ASKLAYOUT** subroutine allows you to report the direction in which the bars will be drawn by returning one of the following values in `direction$`:

Types of Bar Layouts

Type$ value	Description
"HORIZONTAL"	Bars oriented horizontally
"VERTICAL"	Bars oriented vertically

By default, the bar direction is set to a value of "VERTICAL". You can use the **SETLAYOUT** subroutine to change the current bar layout setting.

Example: None

Exceptions: None

See also: **SETLAYOUT**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**

ASKLS Subroutine

Library: SGLIB.TRC

Syntax: CALL ASKLS (*numvar*)

Usage:
```
CALL ASKLS (flag)
```

Summary: Reports whether least-squares linear fits will be drawn automatically for subsequent data plots.

Details: The **ASKLS** subroutine is used to report whether or not least-squares linear fits will be drawn automatically for subsequent data plots produced by the **DATAGRAPH**, **ADDDATAGRAPH**, and **MANYDATAGRAPH** subroutines.

If the **ASKLS** subroutine returns `flag` with a value of 1, subsequent calls to the **DATAGRAPH**, **ADDDATAGRAPH**, and **MANYDATAGRAPH** subroutines will automatically display the graph's least-squares linear fit. If it returns `flag` with a value of 0, they won't.

You can use the **SETLS** subroutine to control whether least-squares linear fitting is currently active or inactive.

Example: None

Exceptions: None

See also: **SETLS**, **ADDLSGRAPH**, **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**

ASKTEXT Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL ASKTEXT (*strvar*, *strvar*, *strvar*)

Usage:
```
CALL ASKTEXT (title$, hlabel$, vlabel$)
```

Summary: Reports the values of the title, horizontal label, and vertical label that will be displayed for subsequently drawn charts and graphs.

Details: The **ASKTEXT** subroutine is used to report the values of the title, horizontal label, and vertical label that will be used to label the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, and **MANYDATAGRAPH** subroutines. (These values also apply to the **PIECHART** subroutine, but only the value of the title is used.)

The **ASKTEXT** subroutine returns the value of the title as `title$`, the value of the horizontal label as `hlabel$`, and the value of the vertical label as `vlabel$`.

You may report the value of the title, the horizontal label, or the vertical label individually using the **ASKTITLE**, **ASKHLABEL**, or **ASKVLABEL** subroutines, respectively.

You may use the **SETTEXT** subroutine to set the current values of the title, the horizontal label, and the vertical label.

Example: None

Exceptions: None

See also: **SETTEXT**, **ASKTITLE**, **ASKHLABEL**, **ASKVLABEL**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **PIECHART**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

ASKTITLE Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL ASKTITLE (*strvar*)

Usage:
```
CALL ASKTITLE (title$)
```

Summary: Reports the value of the title which will be displayed for subsequently drawn charts and graphs.

Details: The **ASKTITLE** subroutine is used to report the value of the title that will be used to label the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**, and **PIECHART** subroutines.

The **ASKTITLE** subroutine returns the value of the title as `title$`.

You may report the current values for the title, the horizontal label, and the vertical label simultaneously using the **ASKTEXT** subroutine. Use the **ASKHLABEL** and **ASKVLABEL** subroutines to report the values of the horizontal label and the vertical label, respectively.

You may use the **SETTITLE** subroutine to set the current value of the title.

Example: None

Exceptions: None

See also: **SETTITLE**, **ASKTEXT**, **ASKHLABEL**, **ASKVLABEL**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **PIECHART**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

ASKVLABEL Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL ASKVLABEL (*strvar*)

Usage:
```
CALL ASKVLABEL (vlabel$)
```

Summary: Reports the value of the vertical label which will be displayed for subsequently drawn charts and graphs.

Details: The **ASKVLABEL** subroutine is used to report the value of the vertical label that will be used to label the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, and **MANYDATAGRAPH** subroutines.

The **ASKVLABEL** subroutine returns the value of the vertical label as `vlabel$`.

You may report the current values for the title, the horizontal label, and the vertical label simultaneously using the **ASKTEXT** subroutine. Use the **ASKHLABEL** and **ASKTITLE** subroutines to report the values of the horizontal label and the title, respectively.

You may use the **SETVLABEL** subroutine to set the current value of the vertical label.

Example: None

Exceptions: None

See also: **SETVLABEL**, **ASKTEXT**, **ASKHLABEL**, **ASKTITLE**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **PIECHART**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

BALANCEBARS Subroutine

Library: BGLIB.TRC

Syntax: CALL BALANCEBARS (*numarrarg*, *numarrarg*, *strarrarg, strarrarg, strex*)

strarrarg:: *strarr*
strarr bowlegs

numarrarg:: *numarr*
numarr bowlegs

Usage: `CALL BARCHART (d1(,), d2(,), units$(), legends$(), colors$)`

Summary: Draws a balanced bar chart, setting off d1(,) values on one side of the axis versus d2(,) values on the other.

Details: The **BALANCEBARS** subroutine draws a balanced bar chart in the current logical window, setting off d1(,) values on one side of the axis versus d2(,) values on the other. This is not a particularly common kind of bar chart, but is useful for comparing income versus expenses, etc.

Simply put, it draws a multi-bar chart of d1(,) on the top or right side of the axis, and the same style chart of d2(,) on the bottom or left side of the axis. Neither array may contain any negative values.

The data arrays d1 and d2 are as in the MULTIBAR subroutine, and the units$ and legends$ arrays label both sets of data.

The `units$` array must contain the same number of items as the `data` array. Each element of the `units$` array will be used as a label for the bar associated with the corresponding element of the `data` array.

The value of `colors$` determines the color scheme that will be used to draw the graph. It generally consists of at least three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels. And the third color will be used for the graph's data.

If `colors$` contains four colors, the third color will be used for drawing bars representing positive values, and the fourth color will be used for drawing bars representing negative values. If `colors$` contains more than four colors, the extra colors will not be used. If `colors$` contains fewer than four colors, the last color specified will be used to fill out the remaining colors. If the value of `colors$` is the null string, then the current foreground color is used for the entire graph.

By default, the **BALANCEBARS** subroutine draws the graph with the bars oriented vertically. The y-axis is automatically scaled to fit the data, and the bars are evenly spaced along the x-axis. The labels will appear beneath each bar.

You can change the graph's orientation so that the bars are drawn horizontally by first invoking the **SETLAYOUT** subroutine with the argument "HORIZONTAL". In this situation, the x-axis will be automatically scaled to fit the data, and the bars will be evenly spaced along the y-axis. The labels will appear to the left of each bar.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, BGBar3.TRU, can be found in the directory TBDEMOS:

```
!  BGBar3  Show a simple balanced bar chart of products,
!          with income/expense for last year and this year.

LIBRARY "..\TBLibs\BGLib.trc"

DIM income(4,2), expense(4,2), units$(4), legend$(2)

MAT READ income, expense, units$, legend$
DATA 43,34, 54,63, 33,12, 62,92   ! Incomes
DATA 39,24, 49,52, 17,13, 43,57   ! Expenses
DATA Faucets, Swings, Hoses, Flamingos     ! Units
DATA Last Year, This Year         ! Legend

CALL SetBarType ("over")
CALL SetText ("Income/Expense: Last 2 Years", "", "Thousands")
LET colors$ = "yellow yellow red green"
CALL BalanceBars (income, expense, units$, legend$, colors$)

GET KEY key

END
```

produces a bar chart representing quarterly profits.

Exceptions:

100	Graph's title is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
111	Data and unit arrays don't match for BarChart.
117	Can't handle this graph range: *low* to *high*.
11008	No such color: *color*.

See also **SETBARTYPE**, **SETTEXT**

BARCHART Subroutine

Library: BGLIB.TRC

Syntax: CALL BARCHART (*numarrarg*, *strarrarg*, *strex*)

strarrarg:: *strarr*
strarr bowlegs

numarrarg:: *numarr*
numarr bowlegs

Usage:

```
CALL BARCHART (data(), units$(), colors$)
```

Summary: Draws a simple bar chart of the specified data values, labeled with the specified units and drawn in the specified color scheme.

Details: The **BARCHART** subroutine draws a bar chart in the current logical window.

The bar chart will contain one bar for each element of the `data` array, and the height of each bar will be determined by the value of its corresponding element in the `data` array.

The `units$` array must contain the same number of items as the `data` array. Each element of the `units$` array will be used as a label for the bar associated with the corresponding element of the `data` array.

The value of `colors$` determines the color scheme that will be used to draw the graph. It

generally consists of at least three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels. And the third color will be used for the graph's data.

If `colors$` contains four colors, the third color will be used for drawing bars representing positive values, and the fourth color will be used for drawing bars representing negative values. If `colors$` contains more than four colors, the extra colors will not be used. If `colors$` contains fewer than four colors, the last color specified will be used to fill out the remaining colors. If the value of `colors$` is the null string, then the current foreground color is used for the entire graph.

By default, the **BARCHART** subroutine draws the graph with the bars oriented vertically. The y-axis is automatically scaled to fit the data, and the bars are evenly spaced along the x-axis. The labels will appear beneath each bar.

You can change the graph's orientation so that the bars are drawn horizontally by first invoking the **SETLAYOUT** subroutine with the argument "HORIZONTAL". In this situation, the x-axis will be automatically scaled to fit the data, and the bars will be evenly spaced along the y-axis. The labels will appear to the left of each bar.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, BGBar1.TRU, can be found in the directory TBDEMOS:

```
!  BGBar1   Draw a simple bar chart.

LIBRARY "..\TBLibs\BGLib.trc"

DIM units$(4), data(4)

MAT READ units$, data
DATA Q-1, Q-2, Q-3, Q-4
DATA 498, 322, 395, 430

CALL SetText ("Quarterly Profits", "Quarter", "Thousands")
CALL BarChart (data, units$, "white cyan magenta")

GET KEY key

END
```

produces a bar chart representing quarterly profits.

Exceptions:

100	Graph's title is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
111	Data and unit arrays don't match for BarChart.
117	Can't handle this graph range: *low* to *high*.

11008 No such color: *color*.

See also: **SETTEXT**, **SETLAYOUT**, **MULTIBAR**, **HISTOGRAM**

DATAGRAPH Subroutine

Library: SGLIB.TRC

Syntax: CALL DATAGRAPH (*numarrarg*,, *numarrarg*, *numex*, *numex*, *strex*)

numarrarg:: *numarr*
numarr bowlegs

Usage:
```
CALL DATAGRAPH (x(), y(), pstyle, lstyle, colors$)
```

Summary: Draws a simple line graph of a set of data points.

Details: The **DATAGRAPH** subroutine draws a line graph of the set of data points whose coordinates are represented by the values of the `x` and `y` arrays.

The `x` array contains the points' x-coordinates, and the `y` array contains their y-coordinates. The coordinates in the two arrays are matched according to their subscripts; that is, the elements with subscripts of 1 within both arrays are interpreted as the coordinates of a single point, as are the elements with subscripts of 2, and so on. Thus, the `x` and `y` arrays must have the same upper and lower bounds, or an error will be generated.

Both the x- and y-axes will be scaled automatically by the **DATAGRAPH** subroutine.

The value of `pstyle` determines the point style that will be used to draw the data points which comprise the graph. The allowable values for `pstyle` are summarized in the following table:

Available Point Styles

Value of `pstyle`	Resulting Point
0	No point (display line only)
1	Dot
2	Plus sign
3	Asterisk
4	Circle
5	X
6	Box
7	Up triangle
8	Down triangle
9	Diamond
10	Solid Box
11	Solid up triangle
12	Solid down triangle
13	Solid diamond

The value of `lstyle` determines the line style that will be used to connect the data points which comprise the graph. The allowable values for `lstyle` are summarized in the following table:

Available Line Styles

Value of `lstyle`	Resulting Line
0	No line (display points only)
1	Solid line
2	Dashed line
3	Dotted line
4	Dash-dotted line

The graph is actually composed of a series of line segments connecting the data points. You can suppress the display of the data points by passing a value of 0 in `pstyle`, or you can suppress

the display of the connecting line segments by passing a value of 0 in `lstyle`.

Note that the **DATAGRAPH** subroutine draws and connects the points in the order in which they are stored in the `x` and `y` arrays. If your points are not stored in left to right order, you may wish to use the **SORTPOINTS** subroutine to order the points before passing them to the **DATAGRAPH** subroutine.

The value of `colors$` determines the color scheme that will be used to draw the graph. It generally consists of three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels. And the third color will be used for the graph's data.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, SGData1.TRU, can be found in the directory TBDEMOS:

```
!  SGData1   Average fuel economy for all cars in USA.  Source: EPA.

LIBRARY "..\TBLibs\SGLib.trc"

DIM year(36), mpg(36)

MAT READ year, mpg

DATA 1940, 1945, 1950, 1951, 1952, 1953, 1954, 1955
DATA 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965
DATA 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975
DATA 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983
DATA 15.29,15.03,14.95,14.99,14.67,14.70,14.58,14.53
DATA 14.36,14.40,14.30,14.30,14.28,14.38,14.37,14.26,14.25,14.07
DATA 14.00,13.93,13.79,13.63,13.57,13.57,13.49,13.10,13.43,13.53
DATA 13.72,13.94,14.06,14.29,15.15,15.54,16.25,16.70

CALL SetText ("Fuel Economy - All Cars", "", "MPG")
CALL DataGraph (year, mpg, 9, 1, "red green yellow")

GET KEY key

END
```

produces a graph of the average fuel economy of all new cars produced in each year from 1940 through 1983.

Exceptions:

100	Graph's title is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
110	Data arrays have different bounds in DataGraph

117 Can't handle this graph range: *low* to *high*.
11008 No such color: *color*.

See also: **SETTEXT**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**

FGRAPH Subroutine

Library: SGFUNC.TRC, SGLIB.TRC

Syntax: CALL FGRAPH (*numex*, *numex*, *numex*, *strex*)

Usage:
```
CALL FGRAPH (startx, endx, style, colors$)
```

Summary: Draws a simple line graph of an externally defined function.

Details: The **FGRAPH** subroutine draws a line graph of the function F(x) over the domain `startx` to `endx`.

The function F(x) must be defined external to your main program. That is, it must be defined using a **DEF** statement or a **DEF** structure which appears after the **END** statement. The function you define must be defined over the entire domain specified. If it is not, the **FGRAPH** subroutine may generate an error or draw the graph incorrectly.

The y-axis will be scaled automatically by the **FGRAPH** subroutine.

The value of `style` determines the line style that will be used to connect the data points which comprise the graph. The allowable values for `style` are summarized in the following table:

Available Line Styles

Value of `style`	**Resulting Line**
0	No line (display points only)
1	Solid line
2	Dashed line
3	Dotted line
4	Dash-dotted line

The graph is actually composed of a series of short line segments. You can control the number of line segments used to display a graph with the **SETGRAIN** subroutine. Using more line segments creates a smoother graph, but takes longer to draw.

The value of `colors$` determines the color scheme that will be used to draw the graph. It generally consists of three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels. And the third color will be used for the graph's data.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, SGFunc1.TRU, can be found in the directory TBDEMOS:

```
!  SGFunc1   Graph the function "Sin(x*x)".

LIBRARY "..\TBLibs\SGFunc.trc", "..\TBLibs\SGLib.trc"

CALL SetText ("Sin(x*x)", "X Values", "Y Values")
CALL Fgraph (-pi, pi, 2, "white white magenta")
```

```
GET KEY key

END

DEF F(x) = Sin(x*x)
```

produces a graph of the function $\sin(x^2)$.

Exceptions:

100	Graph's title is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
117	Can't handle this graph range: *low* to *high*.
11008	No such color: *color*.

See also: **SETTEXT**, **SETGRAIN**, **ADDFGRAPH**, **MANYFGRAPH**

HISTOGRAM Subroutine

Library: BGLIB.TRC

Syntax: CALL HISTOGRAM (*numarrarg*, *strex*)

numarrarg:: *numarr*
numarr bowlegs

Usage:

```
CALL HISTOGRAM (data(), colors$)
```

Summary: Draws a simple histogram of the specified data values in the specified color scheme.

Details: The **HISTOGRAM** subroutine draws a simple histogram in the current logical window.

The histogram automatically "groups" similar values from the `data` array and draws one bar per group. The height of each bar reflects the number of members in the associated group.

For instance, if you use the **HISTOGRAM** subroutine to chart students' grades, it might group all those students with grades in the range 80 through 84 and draw a single bar to represent this group of students. The bars will be labeled "75>", "80>", "85>", and so forth. This means that the first bar represents the group of students whose grades are greater than or equal to 75 but less than 80. The second bar represents the group with grades greater than or equal to 80 but less than 85, and so forth.

The value of `colors$` determines the color scheme that will be used to draw the graph. It generally consists of at least three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels. And the third color will be used for the graph's data.

If `colors$` contains more than three colors, the extra colors will not be used. If `colors$` contains fewer than three colors, the last color specified will be used to fill out the remaining colors. If the value of `colors$` is the null string, then the current foreground color is used for the entire graph.

By default, the **HISTOGRAM** subroutine draws the graph with the bars oriented vertically. The y-axis is automatically scaled to fit the data, and the bars are evenly spaced along the x-axis. The labels will appear beneath each bar.

You can change the graph's orientation so that the bars are drawn horizontally by first invoking the **SETLAYOUT** subroutine with the argument "HORIZONTAL". In this situation, the x-axis will be automatically scaled to fit the data, and the bars will be evenly spaced along the y-axis. The labels will appear to the left of each bar.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, BGHisto1.TRU, can be found in the directory TBDEMOS:

```
!  BGHisto1   Draw a simple histogram.

LIBRARY "..\TBLibs\BGLib.trc"

DIM data(30)

MAT READ data
DATA 65, 70, 93, 85, 83, 68, 77, 92, 83, 85
DATA 89, 72, 75, 81, 80, 84, 73, 79, 78, 84
DATA 80, 79, 72, 91, 85, 82, 79, 76, 74, 79

CALL SetText ("Final Grades", "", "# of Students")
CALL Histogram (data, "white cyan magenta")

GET KEY key

END
```

produces a histogram of student grades.

Exceptions:

100	Graph's title is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
111	Data and unit arrays don't match for Histogram.
117	Can't handle this graph range: *low* to *high*.
11008	No such color: *color*.

See also: **SETTEXT**, **SETLAYOUT**, **BARCHART**, **MULTIHIST**

IBEAM Subroutine

Library: BGLIB.TRC

Syntax: CALL IBEAM (*numarrarg*, *numarrarg*, *strarrarg*, *strex*)

strarrarg:: *strarr*
strarr bowlegs

numarrarg:: *numarr*
numarr bowlegs

Usage:

```
CALL IBEAM (high(), low(), units$(), colors$)
```

Summary: Draws an "I-beam" chart of the specified data values, labeled with the specified units and drawn in the specified color scheme.

Details: The **IBEAM** subroutine draws an "I-beam" chart in the current logical window.

The I-beam chart displays ranges of values and will contain one I-beam for each element of the `high` array. The height and position of each I-beam will be determined by the difference between corresponding elements of the `high` and `low` arrays. For this reason, the `high` and `low` arrays must contain the same number of elements.

The `units$` array must contain the same number of items as the `high` and `low` arrays. Each element of the `units$` array will be used as a label for the I-beam associated with the corresponding elements of the `high` and `low` arrays.

The value of `colors$` determines the color scheme that will be used to draw the graph. It generally consists of at least three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels. And the third color will be used for the graph's data.

If `colors$` contains more than three colors, the extra colors will not be used. If `colors$` contains fewer than three colors, the last color specified will be used to fill out the remaining colors. If the value of `colors$` is the null string, then the current foreground color is used for the entire graph.

By default, the **IBEAM** subroutine draws the graph with the I-beams oriented vertically. The y-axis is automatically scaled to fit the data, and the I-beams are evenly spaced along the x-axis. The labels will appear beneath each I-beam.

You can change the graph's orientation so that the I-beams are drawn horizontally by first invoking the **SETLAYOUT** subroutine with the argument "HORIZONTAL". In this situation, the x-axis will be automatically scaled to fit the data, and the I-beams will be evenly spaced along the y-axis. The labels will appear to the left of each I-beam.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, BGIBeam.TRU, can be found in the directory TBDEMOS:

```
! BGIBeam   Show I-beam chart of stock values.

LIBRARY "..\TBLibs\BGLib.trc"

DIM low(5), high(5), units$(5)

MAT READ low, high, units$
DATA 33.1, 33.2, 34.1, 34.1, 33.1
DATA 34.5, 33.9, 36.2, 34.7, 33.9
DATA Mon, Tues, Wed, Thurs, Fri

CALL SetText ("Stock Values", "Day", "Price")
CALL Ibeam (low, high, units$, "magenta white white")

GET KEY key

END
```

produces an I-beam chart representing the daily ranges of a stock's value over a one week period.

Exceptions: 100 Graph's title is too wide.

102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
111	Data and unit arrays don't match for IBeam.
117	Can't handle this graph range: *low* to *high*.
11008	No such color: *color*.

See also: **SETTEXT**, **SETLAYOUT**

MANYDATAGRAPH Subroutine

Library: SGLIB.TRC

Syntax: CALL MANYDATAGRAPH (*numarrarg*,, *numarrarg*, *numex*, *strarrarg*, *strex*)

strarrarg:: *strarr*
strarr bowlegs

numarrarg:: *numarr*
numarr bowlegs

Usage:
```
CALL MANYDATAGRAPH (x(,), y(,), connect, legends$(), colors$)
```

Summary: Draws multiple line graphs of a set of data points.

Details: The **MANYDATAGRAPH** subroutine draws several line graphs within a single frame. Each graph is based upon a set of data points whose coordinates are represented by the values of corresponding rows of the `x` and `y` arrays. For example, the statement:

```
DIM x(3,15), y(3,15)
```

would create the `x` and `y` matrices for a graph with three lines, each composed of fifteen data points.

Each row of the `x` matrix contains the x-coordinates for the points of a single line graph, and the corresponding row of the `y` matrix contains their y-coordinates. The coordinates in the separate rows of the two matrices are matched according to their second subscripts, or column numbers; that is, the elements with second subscripts of 1 within corresponding rows of both matrices are interpreted as the coordinates of a single point, as are the elements with second subscripts of 2, and so on. Thus, the `x` and `y` matrices must have the same upper and lower bounds in both dimensions, or an error will be generated.

Both the x- and y-axes will be scaled automatically by the **MANYDATAGRAPH** subroutine.

Each graph will use a different point style. These point styles will be drawn in order from the available point styles (with point styles 0 and 1 excepted). When the possible point styles are exhausted, they will be reused from the beginning of the list. For an ordered list of the available point styles, see the discussion of the **DATAGRAPH** subroutine.

If the value of `connect` is not equal to 0, the data points of each line graph will be connected by a line segment.

Note that the **MANYDATAGRAPH** subroutine draws and connects the points in the order in which they are stored in the `x` and `y` matrices. If your points are not stored in left to right order, you may wish to use the **SORTPOINTS2** subroutine to order the points before passing them to the **MANYDATAGRAPH** subroutine.

The **MANYDATAGRAPH** subroutine creates a legend just below the graph's title to assist the user in identifying the various lines. Each label for the legend will be taken from the corresponding element of the `legends$` array. Thus, the number of rows in the `x` and `y` arrays must be equal to the number of elements in the `legends$` array.

If you would like to omit the legend entirely, then pass a `legends$` array which contains no elements.

The value of `colors$` determines the color scheme that will be used to draw the graphs. It generally consists of at least three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels. And the remaining colors will be used for the graphs' data.

If the number of graphs exceeds the number of colors provided for the graphs' data, the **MANYDATAGRAPH** subroutine uses line styles to help distinguish the lines of the graphs. First, it draws solid lines in the colors specified. Then it switches to dashed, dotted, and finally dash-dotted lines. Thus, if you graph five functions with the **MANYFGRAPH** subroutine using the color scheme `"red yellow green blue"` you will get (in order): a solid green line, a solid blue line, a dashed green line, a dashed blue line, and a dotted green line.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, SGData3.TRU, can be found in the directory TBDEMOS:

```
! SGData3   Display multiple sets of data points.

LIBRARY "..\TBLibs\SGLib.trc"

DIM x(5,10), y(5,10), legends$(5)

MAT READ legends$
DATA A, B, C, D, E

FOR i = 1 to 5
    FOR j = 1 to 10
        LET x(i,j) = j
        LET y(i,j) = (i*i*j) ^ 2
    NEXT j
NEXT i

CALL SetText ("Multiple Sets of Data", "Signal", "Reflection")
CALL SetGraphType ("logy")
LET colors$ = "white white magenta cyan"
CALL ManyDataGraph (x, y, 1, legends$, colors$)

GET KEY key

END
```

produces a graph several related data sets.

Exceptions:

100	Graph's title is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.

109 Horizontal marks aren't wide enough—use SetHMarkLen.
110 Data arrays have different bounds in DataGraph
117 Can't handle this graph range: *low* to *high*.
11008 No such color: *color*.

See also: **SETTEXT**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**

MANYFGRAPH Subroutine

Library: SGFUNC.TRC, SGLIB.TRC

Syntax: CALL MANYFGRAPH (*numex*, *numex*, *numex*, *strarr*, *strex*)

Usage:
```
CALL MANYFGRAPH (startx, endx, n, legends$(), colors$)
```

Summary: Draws multiple line graphs based upon an externally defined function.

Details: The **MANYFGRAPH** subroutine draws several line graphs within a single frame. All of the functions drawn are based upon the definition of the function F(x) over the domain `startx` to `endx`. The number of graphs which are to be drawn is indicated by the value of `n`.

The function F(x) must be defined external to your main program. That is, it must be defined using a **DEF** statement or a **DEF** structure which appears after the **END** statement. The functions you define must be defined over the entire domain specified. If they are not, the **MANYFGRAPH** subroutine may generate an error or draw one or more of the graphs incorrectly.

The **MANYFGRAPH** subroutine uses the public variable `fnum` to inform your defined function F(x) which value to compute. The **MANYFGRAPH** subroutine sets the value of `fnum` to 1 when plotting the first function, 2 when plotting the second function, and so on until the number of functions specified by `n` have been plotted. Your defined function F(x) should contain a **PUBLIC** statement listing `fnum` so that the **MANYFGRAPH** subroutine can communicate with it properly. (See the following example for an illustration.)

The y-axis will be scaled automatically by the **MANYFGRAPH** subroutine.

The **MANYFGRAPH** subroutine creates a legend just below the graph's title to assist the user in identifying the various lines. Each label for the legend will be taken from the corresponding element of the `legends$` array. Thus, the value of `n` must be equal to the number of elements in the `legends$` array.

If you would like to omit the legend entirely, then pass a `legends$` array which contains no elements.

The value of `colors$` determines the color scheme that will be used to draw the graphs. It generally consists of at least three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels. And the remaining colors will be used for the graphs' data.

If the number of graphs (represented by the value of `n`) exceeds the number of colors provided for the graphs' data, the **MANYFGRAPH** subroutine uses line styles to help distinguish the lines of the graphs. First, it draws solid lines in the colors specified. Then it switches to dashed, dotted, and finally dash-dotted lines. Thus, if you graph five functions with the **MANYFGRAPH** subroutine using the color scheme `"red yellow green blue"` you will get (in order): a solid green line, a solid blue line, a dashed green line, a dashed blue line, and a dotted green line.

Each graph is actually composed of a series of short line segments. You can control the number of line segments used to display the graphs with the **SETGRAIN** subroutine. Using more line segments creates smoother graphs, but they take longer to draw.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, SGFunc3.TRU, can be found in the directory TBDEMOS:

```
!  SGFunc3   Graph many functions.

LIBRARY "..\TBLibs\SGFunc.trc", "..\TBLibs\SGLib.trc"

DIM legend$(3)

MAT READ legend$
DATA #1, #2, #3

CALL SetText ("Various Waves", "X Values", "Y Values")
LET colors$ = "white white cyan magenta white"
CALL ManyFgraph (-pi, 2*pi, 3, legend$, colors$)

GET KEY key

END

DEF F(x)
    PUBLIC fnum
    SELECT CASE fnum
    CASE 1
         LET F = Sin(x)
    CASE 2
         LET F = 1.5 * Cos(x*2)
    CASE 3
         LET F = .5 * Cos(x+pi/2)
    END SELECT
END DEF
```

produces a single graph of three different functions. Notice the use of the public variable `fnum` to define three distinct behaviors for the single function F(x).

Exceptions:

100	Graph's title is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
112	Data and legend arrays don't match for ManyFGraph.
117	Can't handle this graph range: *low* to *high*.
11008	No such color: *color*.

See also: **SETTEXT**, **SETGRAIN**, **FGRAPH**, **ADDFGRAPH**

MULTIBAR Subroutine

Library: BGLIB.TRC

Syntax: CALL MULTIBAR (*numarrarg*, *strarrarg*, *strarrarg*, *strex*)

strarrarg:: *strarr*
strarr bowlegs

numarrarg:: *numarr*
numarr bowlegs

Usage: `CALL MULTIBAR (data(), units$(), legends$(), colors$)`

Summary: Draws a multi-bar chart of the specified data values, labeled with the specified units and legend and drawn in the specified color scheme.

Details: The **MULTIBAR** subroutine draws a multi-bar chart in the current logical window. In a multi-bar chart, each unit is represented by a cluster of bars. To produce simple bar charts with only one bar per unit, use the **BARCHART** subroutine.

The multi-bar chart will contain one cluster of bars for each row of the `data` array, and each cluster will contain one bar for each column of the `data` array. The height of each bar will be determined by the value of the appropriate element in the `data` array.

For example, if the `data` array contains five rows and three columns, the multi-bar chart will consist of five clusters, and each cluster will contain three bars.

The `units$` array must contain the same number of items as the first dimension of the `data` array. Each element of the `units$` array will be used as a label for the cluster of bars associated with the corresponding row of the `data` array.

The `legends$` array generally must contain the same number of items as the second dimension of the `data` array. The `legends$` array will be used to add a legend to the graph (positioned between the title and the graph itself) which will allow the user to identify the individual bars within the clusters. Each element of the `legends$` array provides the label for the corresponding column of the `data` array. To suppress the appearance of such a legend, pass a `legends$` array which contains zero elements.

The value of `colors$` determines the color scheme that will be used to draw the graph. It generally consists of at least three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels and the legend text. And the third color will be used for the graph's data.

If `colors$` contains more than three colors, the third and following colors will be used in repeating sequence for drawing the bars in each cluster. If `colors$` contains fewer than three colors, the last color specified will be used to fill out the remaining colors. If the value of `colors$` is the null string, then the current foreground color is used for the entire graph.

By default, the **MULTIBAR** subroutine draws the graph with the bars oriented vertically. The y-axis is automatically scaled to fit the data, and the clusters are evenly spaced along the x-axis. The labels stored in the `units$` array will appear beneath each cluster.

You can change the graph's orientation so that the bars are drawn horizontally by first invoking the **SETLAYOUT** subroutine with the argument "HORIZONTAL". In this situation, the x-axis will be automatically scaled to fit the data, and the clusters will be evenly spaced along the y-axis. The labels stored in the `units$` array will appear to the left of each cluster.

By default, the **MULTIBAR** subroutine draws the bars in each cluster side-by-side; however,

they can also be drawn stacked or overlapped. Invoke the **SETBARTYPE** subroutine with an appropriate argument prior to invoking the **MULTIBAR** subroutine in order to determine the arrangement of the bars.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, BGBar2.TRU, can be found in the directory TBDEMOS:

```
!  BGBar2   Draw a simple multi-bar graph.

!  Last year's sales in yellow; this year's in green.

LIBRARY "..\TBLibs\BGLib.trc"

DIM data(4,2), units$(4), legend$(2)

MAT READ data, units$, legend$
DATA 103,106, 47,68, 112,115, 87,94
DATA Books, Software, Cards, Candy
DATA Last Year, This Year

CALL SetBarType ("side")
CALL SetLayout ("h")
CALL SetGrid ("v")
CALL SetText ("Sales: Last Year and Current",
"Thousands","Category")

CALL MultiBar (data, units$, legend$, "red red yellow green")

GET KEY key

END
```

produces a horizontal multi-bar chart representing a comparison of annual sales.

Exceptions:

100	Graph's title is too wide.
101	Graph's legend is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
111	Data and unit arrays don't match for MultiBar.
112	Data and legend arrays don't match for MultiBar.
117	Can't handle this graph range: *low* to *high*.
11008	No such color: *color*.

See also: **SETTEXT**, **SETLAYOUT**, **SETBARTYPE**, **BARCHART**, **HISTOGRAM**

MULTIHIST Subroutine

Library: BGLIB.TRC

Syntax: CALL MULTIHIST (*numarrarg*, *strarrarg*, *strex*)

strarrarg:: *strarr*
strarr bowlegs

numarrarg:: *numarr*
numarr bowlegs

Usage:

```
CALL MULTIHIST (data(), legends$(), colors$)
```

Summary: Draws multiple histograms of the specified data values in a single frame in the specified color scheme.

Details: The **MULTIHIST** subroutine draws multiple histograms in the current logical window. All histograms drawn by the **MULTIHIST** subroutine are overlaid in the same frame, with the bars for similar data values forming "clusters." To produce a simple histogram with only one bar per unit, use the **HISTOGRAM** subroutine.

Each histogram automatically "groups" similar values from a single row of the `data` array and draws one bar per group. Thus, each cluster will contain one bar for each row of the `data` array. The height of each bar reflects the number of members in the associated group.

For instance, if you use the **HISTOGRAM** subroutine to chart students' grades for two different classes, it might group all those students in the first class with grades in the range 80 through 84 and draw a single bar to represent this group of students. When the histogram for the second class was compiled, a bar representing the number of students in that class with grades in the range 80 through 84 would be added to the cluster containing the previous bar. The resulting clusters will be labeled "75>", "80>", "85>", and so forth. This means that the first cluster will contain one bar representing the group of students in the first class whose grades are greater than or equal to 75 but less than 80 and another bar representing students from the second class whose grades fall in the same range. The second cluster will contain bars representing the groups with grades greater than or equal to 80 but less than 85, and so forth.

The `legends$` array generally must contain the same number of items as the second dimension of the `data` array. The `legends$` array will be used to add a legend to the graph (positioned between the title and the graph itself) which will allow the user to identify the individual bars within the clusters. Each element of the `legends$` array provides a label for one of the histograms produced from the `data` array. To suppress the appearance of such a legend, pass a `legends$` array which contains zero elements.

The value of `colors$` determines the color scheme that will be used to draw the graph. It generally consists of at least three color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame, including the horizontal and vertical labels and the legend text. And the third color will be used for the graph's data.

If `colors$` contains more than three colors, the third and following colors will be used in repeating sequence for drawing the bars in each cluster. If `colors$` contains fewer than three colors, the last color specified will be used to fill out the remaining colors. If the value of `colors$` is the null string, then the current foreground color is used for the entire graph.

By default, the **MULTIHIST** subroutine draws the graph with the bars oriented vertically. The y-axis is automatically scaled to fit the data, and the clusters are evenly spaced along the x-axis. The cluster labels will appear beneath each cluster.

You can change the graph's orientation so that the bars are drawn horizontally by first invoking the **SETLAYOUT** subroutine with the argument "HORIZONTAL". In this situation, the x-axis will be automatically scaled to fit the data, and the clusters will be evenly spaced along the y-axis. The cluster labels will appear to the left of each cluster.

By default, the **MULTIHIST** subroutine draws the bars in each cluster side-by-side; however, they can also be drawn stacked or overlapped. Invoke the **SETBARTYPE** subroutine with an appropriate argument prior to invoking the **MULTIHIST** subroutine in order to determine the arrangement of the bars.

The text used for the graph's title and vertical and horizontal labels will be the values most recently set by the **SETTEXT** subroutine.

Example: The following program, BGHisto2.TRU, can be found in the directory TBDEMOS:

```
!  BGHisto2   Draw a multiple histogram.

LIBRARY "..\TBLibs\BGLib.trc"

DIM data(2,30), legend$(2)

MAT READ data, legend$
DATA 65, 70, 93, 85, 83, 68, 77, 92, 83, 85
DATA 89, 72, 75, 81, 80, 84, 73, 79, 78, 84
DATA 80, 79, 72, 91, 85, 82, 79, 76, 74, 79
DATA 75, 60, 83, 75, 73, 88, 67, 82, 73, 75
DATA 79, 62, 65, 71, 70, 74, 63, 69, 68, 74
DATA 70, 69, 62, 81, 75, 72, 69, 66, 64, 69

DATA Day, Evening

CALL SetBarType ("over")
CALL SetText ("Final Grades", "", "# of Students")
CALL MultiHist (data, legend$, "white cyan magenta cyan")

GET KEY key

END
```

produces a horizontal multi-bar chart representing a comparison of annual sales.

Exceptions:

100	Graph's title is too wide.
101	Graph's legend is too wide.
102	Graph's horizontal label is too wide.
103	Graph's vertical label is too long.
104	Need more room for graph's vertical marks.
105	Need more room for graph's horizontal marks.
106	Need greater width for graph.
107	Need greater height for graph.
108	Vertical marks aren't wide enough—use SetVMarkLen.
109	Horizontal marks aren't wide enough—use SetHMarkLen.
111	Data and unit arrays don't match for MultiHist.
112	Data and legend arrays don't match for MultiHist.
117	Can't handle this graph range: *low* to *high*.
11008	No such color: *color*.

See also: **SETTEXT**, **SETLAYOUT**, **SETBARTYPE**, **HISTOGRAM**, **BARCHART**

PIECHART Subroutine

Library: BGLIB.TRC

Syntax: CALL PIECHART (*numarrarg*, *strarrarg*, *strex*, *numex*, *numex*)

strarrarg:: *strarr*
strarr bowlegs

numarrarg:: *numarr*
numarr bowlegs

Usage:

```
CALL PIECHART (data(), units$(), colors$, wedge, percent)
```

Summary: Draws a pie chart of the specified data values, labeled with the specified units and drawn in the specified color scheme.

Details: The **PIECHART** subroutine draws a pie chart in the current logical window.

A pie chart is displayed as a circle divided into wedges. The pie chart will contain one wedge for each element of the `data` array, and the proportion of the circle's area allocated to each wedge will be determined by the proportional relationship of the value of its corresponding element in the `data` array to the sum of the elements of the `data` array.

The wedge associated with the first element of the `data` array is placed at the top of the pie, and the remaining items of the `data` array are arranged in order clockwise around the remaining portion of the pie.

The `units$` array must contain the same number of items as the `data` array. Each element of the `units$` array will be used as a label for the wedge of the pie associated with the corresponding element of the `data` array. Each label will be connected to its associated wedge by a line. If an element of the `units$` array has a value of the null string, the associated wedge will have neither a label nor a connecting line.

The value of `colors$` determines the color scheme that will be used to draw the graph. It generally consists of at least four color names (in any combination of uppercase or lowercase letters) separated by spaces. The valid color names are:

RED	MAGENTA	YELLOW
GREEN	BLUE	CYAN
BROWN	WHITE	BLACK
	BACKGROUND	

The value of `colors$` may also contain color numbers instead of color names, allowing you to access any of the colors supported by the current computer system.

The first color specified by the value of `colors$` will be used for the graph's title. The second color will be used for the graph's frame. And the remaining colors will be used repeatedly for the wedges of the pie.

If the value of `wedge` fall between the lower and upper bounds of the `data` array, inclusive, the wedge of the pie associated with the element of `data` whose index is represented by the value of `wedge` will be exploded out of the pie. That is, it will be drawn slightly separated from the rest of the pie in order to draw the user's attention. If the value of `wedge` falls outside this range, no wedge will be exploded out of the pie.

If the value of `percent` is non-zero, each wedge will be labeled not only with the corresponding element of the `units$` array, but also with the percentage of the total which it represents. If the value of `percent` is 0, the wedges will be labeled only with the elements of the `units$` array. Note that the percentages are rounded before being displayed. Therefore, it is not guaranteed that they will add up to exactly 100%.

Example: The following program, BGPie.TRU, can be found in the directory TBDEMOS:

```
!  BGPie   Draw a simple pie chart.

!  Highlight hammers, and show percentages.

LIBRARY "..\TBLibs\BGLib.trc"

DIM data(5), units$(5)

MAT READ data, units$
DATA 120, 34, 87, 65, 21
DATA Nails, Hammers, Saws, Pliers, Awls

CALL SetTitle ("Honest Boy (tm) Product Income")
CALL PieChart (data, units$, "yellow green red", 2, 1)

GET KEY key

END
```

produces a pie chart representing income by product, highlighting hammers and displaying percentages with each label.

Exceptions:

100	Graph's title is too wide.
106	Need greater width for graph.
107	Need greater height for graph.
111	Data and unit arrays don't match for PieChart.
11008	No such color: *color*.

See also: **SETTITLE**

SETANGLE Subroutine

Library: SGLIB.TRC

Syntax: CALL SETANGLE (*strex*)

Usage:
```
CALL SETANGLE (measure$)
```

Summary: Controls the manner in which subsequent polar graphs drawn by the various data and function plotting subroutines will interpret angle measurements.

Details: The **SETANGLE** subroutine is used to control the manner in which subsequent data and function polar plots produced by the **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**, **ADDFGRAPH**, and **MANYFGRAPH** subroutines will interpret angle measurements.

When these subroutines interpret angle measurements, they interpret them as radians by default. However, by passing a value of "DEG" as `measure$`, you can instruct them to interpret angles in degrees. Passing a value of "RAD" to the **SETANGLE** subroutine will reset the default interpretation.

Note that the **SETANGLE** subroutine only controls the interpretation of angular coordinates by polar graphs. Use the **SETGRAPHTYPE** subroutine to cause subsequent graphs to be drawn as polar graphs.

You can use the **ASKANGLE** subroutine to determine the manner in which the next data or function polar plot will interpret angular coordinates.

Example: None

Exceptions: None

See also: **ASKANGLE**, **SETGRAPHTYPE**, **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**, **ADDFGRAPH**, **MANYFGRAPH**

SETBARTYPE Subroutine

Library: BGLIB.TRC

Syntax: CALL SETBARTYPE (*strex*)

Usage:
```
CALL SETBARTYPE (type$)
```

Summary: Controls the arrangement of the bars within each group of a multiple bar chart or histogram.

Details: The **SETBARTYPE** subroutine is used to control the arrangement of the bars within each group of a bar chart or histogram produced by a subsequent invocation of the **MULTIBAR** or **MULTIHIST** subroutine.

Both the **MULTIBAR** and **MULTIHIST** subroutines draw multiple bar-based graphs in a single frame. In such a graph, bars associated with a particular unit are grouped together.

The **SETBARTYPE** subroutine allows you to control how the bars in each group will be arranged by passing one of the following values in `type$`:

Types of Bar Groupings

`Type$` **value**	**Description**
`"SIDE"`	Bars arranged side by side with space between them
`"STACK"`	Bars stacked one above the other
`"OVER"`	Bars arranged side by side but overlapped slightly

The value of `type$` may be specified in any combination of uppercase and lowercase letters.

If the value of `type$` does not represent one of these values, an error will be generated.

By default, the bar type is set to a value of `"SIDE"`. You can use the **ASKBARTYPE** subroutine to report the current bar type setting.

Example: See the example programs in the desciptions of BALANCEBARS (BGBar3.TRU,) MULTIBAR (BGBar2.TRU,) and MULTIHIST (BGHisto2.TRU) for examples of the use of this subroutine.

Exceptions: 130 No such barchart type: *xxx*

See also: **ASKBARTYPE**, **MULTIBAR**, **MULTIHIST**

SETGRAIN Subroutine

Library: SGLIB.TRC

Syntax: CALL SETGRAIN (*numex*)

Usage: `CALL SETGRAIN (grain)`

Summary: Controls the grain with which subsequent invocations of the various function plotting subroutines will draw the line graph.

Details: The **SETGRAIN** subroutine controls the grain with which subsequent invocations of the **FGRAPH**, **ADDFGRAPH**, and **MANYFGRAPH** subroutines will draw the line representing the function.

These subroutines actually graph the curve of the function which they are plotting as a series of line segments. The ***grain*** controls the number of line segments used to form each graphed curve. The higher the value of the grain, the more line segments are used and the smoother the resulting curve appears. However, higher grains also mean more work for the computer, and this means that each curve takes longer to draw.

By default, the **FGRAPH**, **ADDFGRAPH**, and **MANYFGRAPH** subroutines use a grain value of 64, which means that each line graph is composed of 64 individual line segments. This value strikes a generally acceptable balance of smoothness and speed, but you can change this value by passing the new grain value in the `grain` argument to the **SETGRAIN** subroutine.

You can use the **ASKGRAIN** subroutine to report the current grain value.

Example: The following program, SGGrain.TRU, can be found in the directory TBDEMOS:

```
!  SGGrain   Demonstrate SetGrain.

LIBRARY "..\TBLibs\SGFunc.trc", "..\TBLibs\SGLib.trc"

OPEN #1: screen 0, .49, 0, 1

CALL SetGrain (10)
CALL SetTitle ("Grain = 10")
CALL Fgraph (-pi, pi, 1, "white white magenta")

OPEN #2: screen .5, 1, 0, 1

CALL SetGrain (100)
CALL SetTitle ("Grain = 100")
CALL Fgraph (-pi, pi, 1, "white white magenta")

GET KEY key

END

DEF F(x) = Sin(3*x)
```

demonstrates the use of the **SETGRAIN** subroutine by displaying two graphs of the same function side by side — one with a grain of 10 and the other with a grain of 100.

Exceptions: None

See also: **ASKGRAIN**, **FGRAPH**, **ADDFGRAPH**, **MANYFGRAPH**

SETGRAPHTYPE Subroutine

Library: SGLIB.TRC

Syntax: CALL SETGRAPHTYPE (*strex*)

Usage:
```
CALL SETGRAPHTYPE (type$)
```

Summary: Controls the type of graph that will be drawn by subsequent data and function plotting subroutines.

Details: The **SETGRAPHTYPE** subroutine is used to control the type of graph that will be produced for subsequent data and function plots produced by the **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**, **ADDFGRAPH**, and **MANYFGRAPH** subroutines.

The type of subsequent graphs is determined by the value passed as `type$`. The possible values of `type$` are:

Types of Graphs

`Type$` **value**	**Description**
`"XY"`	Normal graph
`"LOGX"`	Semi-logarithmic graph with x-axis logarithmically scaled
`"LOGY"`	Semi-logarithmic graph with y-axis logarithmically scaled
`"LOGXY"`	Logarithmic graph with both x- and y-axes logarithmically scaled
`"POLAR"`	Polar graph

Logarithmic and semi-logarithmic graphs look very similar to normal graphs, but one or both of the axes is scaled logarithmically.

Polar graphs, however, look quite different from normal graphs in that they are circular. For this reason, the horizontal and vertical labels are ignored for polar graphs; only the title is shown.

When a graphing routine is used to draw a polar graph, what would normally be the x- and y-coordinates are interpreted as r and theta (or distance and angle) coordinates, respectively. Therefore, as you might expect, the function plotting subroutines expect to find an externally defined function in the form `r = F(theta)`.

Polar graphs interpret angle measures as radians by default, but you can change this interpretation using the **SETANGLE** subroutine.

You can use the **ASKGRAPHTYPE** subroutine to determine the type of graph that will be used for the next data or function plot.

Example: See the example program in the description of MANYDATAGRAPH (SGData3.TRU) for an example of the use of this subroutine.

Exceptions: None

See also: **ASKGRAPHTYPE**, **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**, **FGRAPH**, **ADDFGRAPH**, **MANYFGRAPH**

SETGRID Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL SETGRID (*strex*)

Usage:
```
CALL SETGRID (style$)
```

Summary: Controls the presence, direction, and type of the grid within subsequently drawn charts and graphs.

Details: The **SETGRID** subroutine is used to control the presence, direction, and type of the grid within the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH** subroutines.

The **SETGRID** subroutine allows you to control the presence and direction of the grid lines by passing one of the following values in `style$`:

Available Grid Directions

Style$ value	Description
""	No grid lines
"H"	Horizontal grid lines only
"V"	Vertical grid lines only
"HV"	Both horizontal and vertical grid lines

The value of `style$` may be specified in any combination of uppercase and lowercase letters. In addition, the value of `style$` may include instructions that indicate the type of grid lines that you would like drawn. By default, grid lines are drawn as solid lines. However, you can append one of the following modifiers to a letter in the value of `style$` to specify a different line type for grid lines traveling in that direction:

Available Grid Type Modifiers

Modifier	Description
-	Dashed grid lines
.	Dotted grid lines
-.	Dash-dotted grid lines

For example, passing a value of `"H-.V"` for `style$` would result in dash-dotted grid lines in the horizontal direction and solid grid lines in the vertical direction.

If the value of `type$` does not represent a valid value, however, an error will be generated.

By default, the grid lines are turned off. You can use the **ASKGRID** subroutine to report the current grid setting.

Example: See the example program in the description of MULTIBAR (BGBar2.TRU) for an example of the use of this subroutine.

Exceptions: 113 No such SetGrid direction: *xxx*

See also: **ASKGRID**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

SETHLABEL Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL SETHLABEL (*strex*)

Usage:
```
CALL SETHLABEL (hlabel$)
```

Summary: Sets the value of the horizontal label which will be displayed for subsequently drawn charts and graphs.

Details: The **SETHLABEL** subroutine is used to set the value of the horizontal label that will be used to label the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, and **MANYDATAGRAPH** subroutines.

The **SETHLABEL** subroutine expects the value of the horizontal label to be passed as `hlabel$`. Passing a null string effectively eliminates the horizontal label.

If the value you set for the horizontal label exceeds the available room, the graphing subroutine which draws the next graph will generate an error.

There is no default value for the horizontal label. Therefore, if you want it to appear, you will need to specify its values before drawing the graph.

You may specify new values for the title, the horizontal label, and the vertical label simultaneously using the **SETTEXT** subroutine. Use the **SETVLABEL** and **SETTITLE** subroutines to set the values of the vertical label and the title, respectively.

You may use the **ASKHLABEL** subroutine to report the current value of the horizontal label.

Example: None

Exceptions: None

See also: **ASKHLABEL**, **SETTEXT**, **SETVLABEL**, **SETTITLE**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **PIECHART**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

SETLAYOUT Subroutine

Library: BGLIB.TRC

Syntax: CALL SETLAYOUT (*strex*)

Usage:
```
CALL SETLAYOUT (direction$)
```

Summary: Controls the direction of the bars within subsequently drawn bar charts and histograms.

Details: The **SETLAYOUT** subroutine is used to control the direction of the bars within each bar chart or histogram produced by a subsequent invocation of the **MULTIBAR** or **MULTIHIST** subroutine.

The **SETLAYOUT** subroutine allows you to control the direction in which the bars will be drawn by passing one of the following values in `direction$`:

Types of Bar Groupings

`Direction$` **value**	**Description**
`"HORIZONTAL"`	Bars oriented horizontally
`"VERTICAL"`	Bars oriented vertically

The value of `type$` may be specified in any combination of uppercase and lowercase letters. In addition, the value of `type$` may be truncated to any number of letters. That is, values of `"H"` and `"V"` will suffice. If the value of `type$` does not represent a valid value, however, an error will be generated.

By default, the bar direction is set to a value of `"VERTICAL"`. You can use the **ASKLAYOUT** subroutine to report the current bar layout setting.

Example: See the example program in the description of MULTIBAR (BGBar2.TRU) for an example of the use of this subroutine.

Exceptions: 131 No such barchart direction: *xxx*

See also: **ASKLAYOUT**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**

SETLS Subroutine

Library: SGLIB.TRC

Syntax: CALL SETLS (*numex*)

Usage:
```
CALL SETLS (flag)
```

Summary: Controls whether least-squares linear fits will be drawn automatically for subsequent data plots.

Details: The **SETLS** subroutine is used to control whether or not least-squares linear fits will be drawn automatically for subsequent data plots produced by the **DATAGRAPH**, **ADDDATAGRAPH**, and **MANYDATAGRAPH** subroutines.

The least-squares linear fit of a data plot is the straight line which best fits the locations of the data points. That is, the least-squares linear fit of a data plot is the straight line which minimizes the vertical distance between itself and each of the data points which form the plot. Such a line may be used to help predict where data points might lie in areas of the graph for which data is unavailable.

By default, the **DATAGRAPH**, **ADDDATAGRAPH**, and **MANYDATAGRAPH** subroutines draw the data plots without displaying the least-squares linear fit of the data points. However, invoking the **SETLS** subroutine with the value of `flag` equal to 1 will instruct subsequent invocations of these routines to add such a linear fit to each graph they draw. You may then turn off this line fitting by invoking the **SETLS** subroutine again with the value of `flag` equal to 0.

When the **DATAGRAPH** and **ADDDATAGRAPH** subroutines draw a linear fit, they draw a solid line in the data color. The **MANYDATAGRAPH** subroutine draws each graph's linear fit in the same color and line style as the lines connecting that graph's data points.

You can use the **ASKLS** subroutine to determine whether least-squares linear fitting is currently active or inactive.

Example: None

Exceptions: None

See also: **ASKLS**, **ADDLSGRAPH**, **DATAGRAPH**, **ADDDATAGRAPH**, **MANYDATAGRAPH**

SETTEXT Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL SETTEXT (*strex*, *strex*, *strex*)

Usage:
```
CALL SETTEXT (title$, hlabel$, vlabel$)
```

Summary: Sets the values of the title, horizontal label, and vertical label which will be displayed for subsequently drawn charts and graphs.

Details: The **SETTEXT** subroutine is used to set the values of the title, horizontal label, and vertical label that will be used to label the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, and **MANYDATAGRAPH** subroutines. (These values also apply to the **PIECHART** subroutine, but only the value of the title is used.)

The **SETTEXT** subroutine expects the value of the title to be passed as `title$`, the value of the horizontal label to be passed as `hlabel$`, and the value of the vertical label to be passed as `vlabel$`. Passing a null string for any of these values effectively eliminates that label.

If the values you set for one or more of these labels exceeds the available room, the graphing subroutine which draws the next graph will generate an error.

There are no default values for the title, the horizontal label, or the vertical label. Therefore, if you want any of them to appear, you will need to specify their values before drawing the graph.

You may specify a new value for the title, the horizontal label, or the vertical label individually using the **SETTITLE**, **SETHLABEL**, or **SETVLABEL** subroutines, respectively.

You may use the **ASKTEXT** subroutine to report the current values of the title, the horizontal label, and the vertical label.

Example: See almost all the example programs described in this section for examples of the use of this subroutine.

Exceptions: None

See also: **ASKTEXT**, **SETTITLE**, **SETHLABEL**, **SETVLABEL**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **PIECHART**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

SETTITLE Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL SETTITLE (*strex*)

Usage:
```
CALL SETTITLE (title$)
```

Summary: Sets the value of the title which will be displayed for subsequently drawn charts and graphs.

Details: The **SETTITLE** subroutine is used to set the value of the title that will be used to label the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**, and **PIECHART** subroutines.

The **SETTITLE** subroutine expects the value of the title to be passed as `title$`. Passing a null string effectively eliminates the title.

If the value you set for the title exceeds the available room, the graphing subroutine which draws the next graph will generate an error.

There is no default value for the title. Therefore, if you want it to appear, you will need to specify its values before drawing the graph.

You may specify new values for the title, the horizontal label, and the vertical label simultaneously using the **SETTEXT** subroutine. Use the **SETHLABEL** and **SETVLABEL** subroutines to set the values of the horizontal label and the vertical label, respectively.

You may use the **ASKTITLE** subroutine to report the current value of the title.

Example: See the examples programs in the descriptions of SETGRAIN (SGGrain.TRU) and SORTPOINTS (SGSortPt.TRU) for examples of the use of this subroutine.

Exceptions: None

See also: **ASKTITLE**, **SETTEXT**, **SETHLABEL**, **SETVLABEL**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **PIECHART**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

SETVLABEL Subroutine

Library: BGLIB.TRC or SGLIB.TRC

Syntax: CALL SETVLABEL (*strex*)

Usage:
```
CALL SETVLABEL (vlabel$)
```

Summary: Sets the value of the vertical label which will be displayed for subsequently drawn charts and graphs.

Details: The **SETVLABEL** subroutine is used to set the value of the vertical label that will be used to label the frame of graphs or charts drawn by subsequent invocations of the **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, and **MANYDATAGRAPH** subroutines.

The **SETVLABEL** subroutine expects the value of the vertical label to be passed as `vlabel$`. Passing a null string effectively eliminates the vertical label.

If the value you set for the vertical label exceeds the available room, the graphing subroutine which draws the next graph will generate an error.

There is no default value for the vertical label. Therefore, if you want it to appear, you will need to specify its values before drawing the graph.

You may specify new values for the title, the horizontal label, and the vertical label simultaneously using the **SETTEXT** subroutine. Use the **SETHLABEL** and **SETTITLE** subroutines to set the values of the horizontal label and the title, respectively.

You may use the **ASKVLABEL** subroutine to report the current value of the vertical label.

Example: None

Exceptions: None

See also: **ASKVLABEL**, **SETTEXT**, **SETHLABEL**, **SETTITLE**, **BARCHART**, **MULTIBAR**, **HISTOGRAM**, **MULTIHIST**, **IBEAM**, **PIECHART**, **FGRAPH**, **MANYFGRAPH**, **DATAGRAPH**, **MANYDATAGRAPH**

SETXSCALE Subroutine

Library: SGLIB.TRC

Syntax: CALL SETXSCALE (*numex, numex*)

Usage:
```
CALL SETXSCALE (70, 170)
```

Summary: Turns off auto-scaling and sets the x-range for subsequent graphs.

Details: The **SETXSCALE** subroutine is used to set the value of the x-scale for subsequent graphs. It turns off auto-scaling. The actual x-range may be slightly different as this subroutine may round to "good-looking" numbers.

Example: None.

Exceptions: None

See also: **SETYSCALE**

SETYSCALE Subroutine

Library: SGLIB.TRC

Syntax: CALL SETYSCALE (*numex, numex*)

Usage:
```
CALL SETYSCALE (70, 170)
```

Summary: Turns off auto-scaling and sets the y-range for subsequent graphs.

Details: The **SETYSCALE** subroutine is used to set the value of the y-scale for subsequent graphs. It turns off auto-scaling. The actual y-range may be slightly different as this subroutine may round to "good-looking" numbers.

Example: See the example in the description of ADDLSGRAPH (SLSquar.TRU) for an example of the use of this subroutine.

Exceptions: None

See also: **ADDLSGRAPH, SETXSCALE**

SORTPOINTS Subroutine

Library: SGLIB.TRC

Syntax: CALL SORTPOINTS (*numarrarg*, *numarrarg*)

numarrarg:: *numarr*
numarr bowlegs

Usage:
```
CALL SORTPOINTS (x(), y())
```

Summary: Sorts the one-dimensional parallel arrays `x` and `y` into ascending order by the values of stored in the `x` array.

Details: The **SORTPOINTS** subroutine sorts the parallel, one-dimensional arrays `x` and `y` into ascending order by the values stored in the `x` array.

Parallel arrays are simply arrays in which elements with identical subscripts are related. For instance, the `x` and `y` arrays are considered to be parallel if the first element of the `x` array is related to the first element of the `y` array, the second to the second, and so forth for each element in both arrays. When parallel arrays are sorted, the elements in both arrays are rearranged in an identical manner so as to maintain these relationships.

The **SORTPOINTS** subroutine is useful for sorting the arrays of coordinates passed into the **DATAGRAPH** and **ADDDATAGRAPH** subroutines, but it can be used to sort any pair of one-dimensional parallel arrays.

To sort two-dimensional arrays in a similar fashion, use the **SORTPOINTS2** subroutine. To sort a single one-dimensional array, use the **SORTN** subroutine.

Example: The following program, SGSortPt.TRU, can be found in the directory TBDEMOS:

```
!  SGSortPt   Display unsorted vs. sorted data points.

LIBRARY "..\TBLibs\SGLib.trc"

DIM x(10), y(10)

FOR i = 1 to 10                    ! Get some unsorted data points
    LET x(i) = rnd
    LET y(i) = rnd
```

```
NEXT i

OPEN #1: screen 0, .49, 0, 1      ! Left: unsorted points
CALL SetTitle ("Unsorted")
CALL DataGraph (x, y, 10, 3, "")

OPEN #2: screen .5, 1, 0, 1       ! Right: sorted points
CALL SetTitle ("Sorted")
CALL SortPoints (x, y)
CALL DataGraph (x, y, 10, 3, "")

GET KEY key

END
```

demonstrates the usefulness of using the **SORTPOINTS** subroutine with the **DATAGRAPH** subroutine.

Exceptions: None

See also: **SORTPOINTS2**, **DATAGRAPH**, **ADDDATAGRAPH**, **SORTN**

SORTPOINTS2 Subroutine

Library: SGLIB.TRC

Syntax: CALL SORTPOINTS2 (*numarrarg*, *numarrarg*)

numarrarg:: *numarr*
numarr bowlegs

Usage: `CALL SORTPOINTS2 (x(,), y(,))`

Summary: Sorts the parallel rows of the two-dimensional arrays `x` and `y` into ascending order by the values of stored in rows of the `x` array.

Details: The **SORTPOINTS2** subroutine sorts the elements of the parallel rows of the two-dimensional arrays `x` and `y` into ascending order by the values stored in the rows of the `x` array.

Parallel arrays are simply arrays in which elements with identical subscripts are related. For instance, the `x` and `y` arrays are considered to be parallel if the first element of the `x` array is related to the first element of the `y` array, the second to the second, and so forth for each element in both arrays. When parallel arrays are sorted, the elements in both arrays are rearranged in an identical manner so as to maintain these relationships.

The **SORTPOINTS2** subroutine treats corresponding rows of the `x` and `y` arrays as individually parallel one-dimensional arrays. That is, the elements of each pair of corresponding rows are rearranged independently of the other rows.

The **SORTPOINTS2** subroutine is useful for sorting the arrays of coordinates passed into the **MANYDATAGRAPH** subroutine, but it can be used to sort any pair of two-dimensional arrays with parallel rows.

To sort one-dimensional arrays in a similar fashion, use the **SORTPOINTS** subroutine. To sort a single one-dimensional array, use the **SORTN** subroutine.

Example: None

Exceptions: None

See also: **SORTPOINTS**, **MANYDATAGRAPH**, **SORTN**