

SOME METHODS TO EXTEND SAS/GRAPH®

Paul L. Marsh, Department of Statistics, North Carolina State University

ABSTRACT

There are many situations where the standard procedures within SAS/GRAPH can not produce the desired results. On the other hand, use of the Data Step Graphics Interface (DSGI) or the ANNOTATE facility can appear formidable to the novice SAS® user.

This talk will present some simple techniques that can be used in a variety of situations to greatly enhance the graphics or report-writing capabilities of standard SAS. Not only will the output be presented, but discussion of the underlying code (often only a few lines) will be part of the talk. Also, the speaker will be available in the conference Coder's Corner if any attendee wants ideas about a specific problem.

INTRODUCTION

How many times has SAS/GRAPH failed to produce exactly the graph you want? For example, you wanted slightly different plotting symbols or text than that offered by the SAS/GRAPH procedures.

SAS/GRAPH is basically a presentation-graphics product. As such, most of the emphasis within the software is visual appearance. As you progress away from the basic appearance of a plot towards illumination and visualization of data, SAS/GRAPH offers fewer and fewer built-in features. Fortunately, some simple techniques can be employed to greatly extend the utility of the SAS/GRAPH procedures. The remainder of this paper is a series of examples that illustrate some of these techniques. Sample output is included in the discussion. Sample code can be found at the end of the paper.

TEXT ON MULTI-GRAPHS

The template facility with PROC GREPLAY makes it easy to place multiple graphs on a single page.

However, what is the easiest way to place a title on that page?

The examples in the SAS/GRAPH manuals lead one away from the simple solution: create a whole page panel within the template definition. For example, say we wish to place four graphs on a single page. Our template definition (TDEF) statement or window might look like this:

```
tdef h2v2 des='4 panels on 1 page'
1/ llx=0 lly=0 lrx=50 lry=0 urx=50 ury=50 ulx=0 ury=50
2/ copy=1 xlatex=50
3/ copy=1 xlatey=50
4/ copy=3 xlatex=50;
```

For those readers unfamiliar with how it operates, the template facility merely defines a scale of 0-100 for each edge of a page. Using these scales, a programmer can subdivide the page into panels by defining the four corners of each panel (llx=lower left x, lly=lower left y, lrx=lower right x, lry=lower right y, urx=upper right x, ury=upper right y, ulx=upper left x, ury=upper right y). For the example above, the line beginning 1/ creates a panel in the lower left hand corner of the page that covers one-half of the page vertically and horizontally. The remainder of that code replicates that panel to the other three quadrants of the page.

This example is much like those found in the SAS/GRAPH manuals. The question is: where are we supposed to put the title? The answer is in two parts. First, shrink the panels a bit vertically in order to leave some room at the top of the page. If we allow each page to cover forty-five percent (45%) of the page vertically while keeping them flush to each other and the bottom of the page, we would now leave the top ten percent (10%) of the page empty.

To effectively use this top of the page, we will define a fifth panel that spans the entire page (see the code below). Yes, panels may overlap, even coincide!!! We define the title panel to cover the entire page so that it is easier to scale the text of the title(s). After all, we

want the titles to pertain to the whole page. Figure 1 contains the four individual graphs that are discussed below (see "WRITING SAS WITH SAS"). See sample program 1 at end of paper for code listing.

```
tdef h2v2 des='4 panels plus text on 1 page'
1/ llx=0 lly=0 lrx=50 lry=0 urx=50 ury=45 ulx=0 ury=45
2/ copy=1 xlatex=50
3/ copy=1 xlatey=45
4/ copy=3 xlatex=50
5/ llx=0 lly=0 lrx=100 lry=0 urx=100 ury=100 ulx=0
ury=100;
```

WRITING SAS WITH SAS

Another trick to keep in mind when building "unordinary" graphs is that the SAS data step is a programming language. Actually this should be kept in

mind whenever using SAS, but the example below tells how to get SAS to write and then execute a set of SAS/GRAPH statements.

A user wishes to produced a set of simple scatterplots of the form $Y \times X = B$ where B will select which SYMBOL statements to use for a particular graph. Unfortunately, SAS/GRAPH does not handle this need directly. You may plot the formatted values of B (assuming B is a nominal variable) or you may specify 1 value after the equals (=) sign that corresponds to a specific SYMBOL statement. If B has multiple levels, you are left to subsetting the data and doing some type of overlaying.

Fortunately, it is not difficult to get SAS to produce and then execute some SAS code. And to finish the process, we will use our friend PROC GREPLAY to

Four Graphs on Page with A Full Title

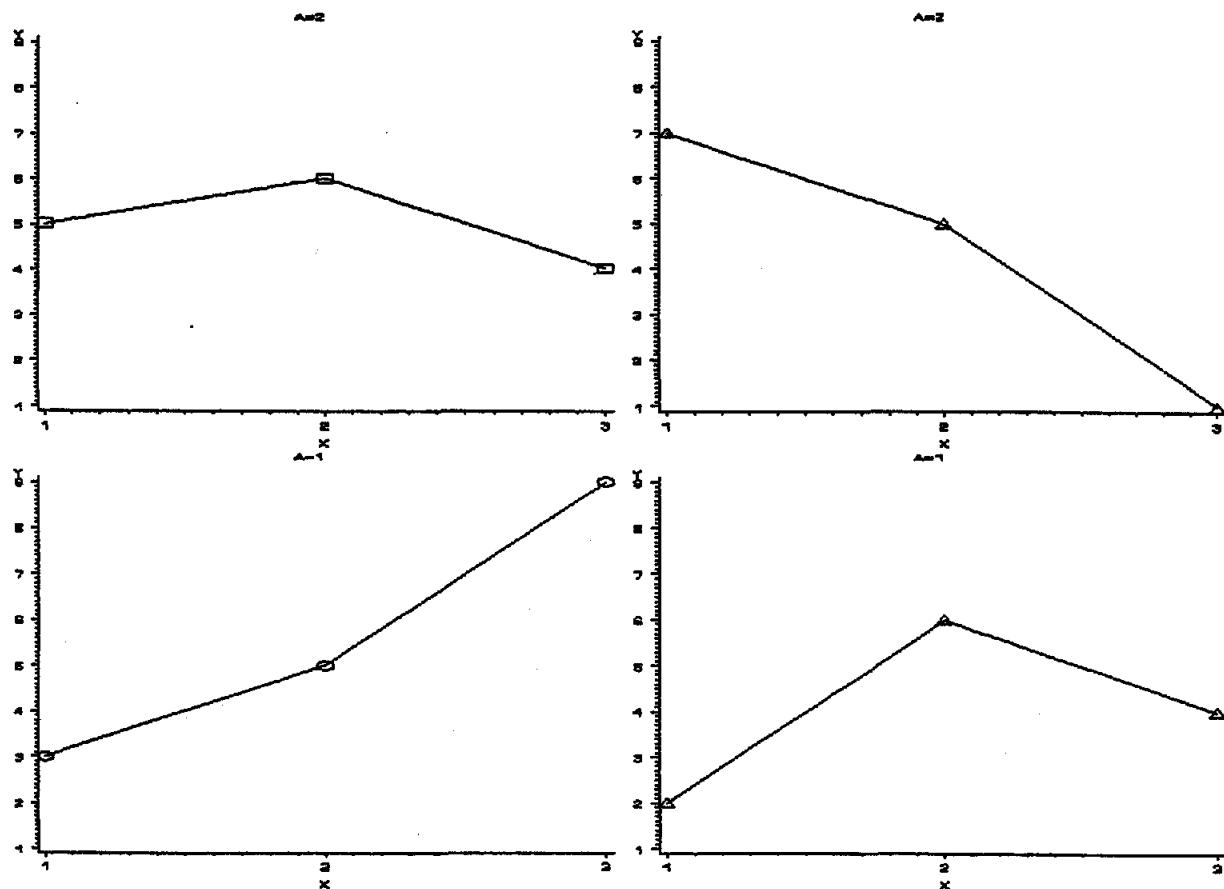


Figure 1. Placement of titles on a PROC GREPLAY template

overlay the plots correctly.

Sample program 1 listed at the end of this paper is a solution to the above problem. Initially, we read in four variables (A, B, X, and Y). X and Y are the horizontal and vertical axis values, respectively. The values of B select the symbol statement for a particular X-Y combination. Variable A is a by-variable; we wish to produce a graph for each of its two values.

We set up the SYMBOL statements for the plot. A common mistake is to omit the C= (color) parameter on those statements. If this parameter is omitted, SAS/GRAPH will cycle through each of the colors for

the current graphics device before attempting to use the parameters supplied on the next SYMBOL statement. Since the available colors varies among graphics devices, your code will potentially produce different graphs as you move to a new device. CAVEAT EMPTOR: Always use the COLOR= parameter!!!

The next two PROC FREQ steps allow us to derive which symbols are needed for each graph and the total number of graphs, respectively. The subsequent PROC MEANS derives the extreme values of X and Y so that we can control the axes and make them uniform across all graphs.

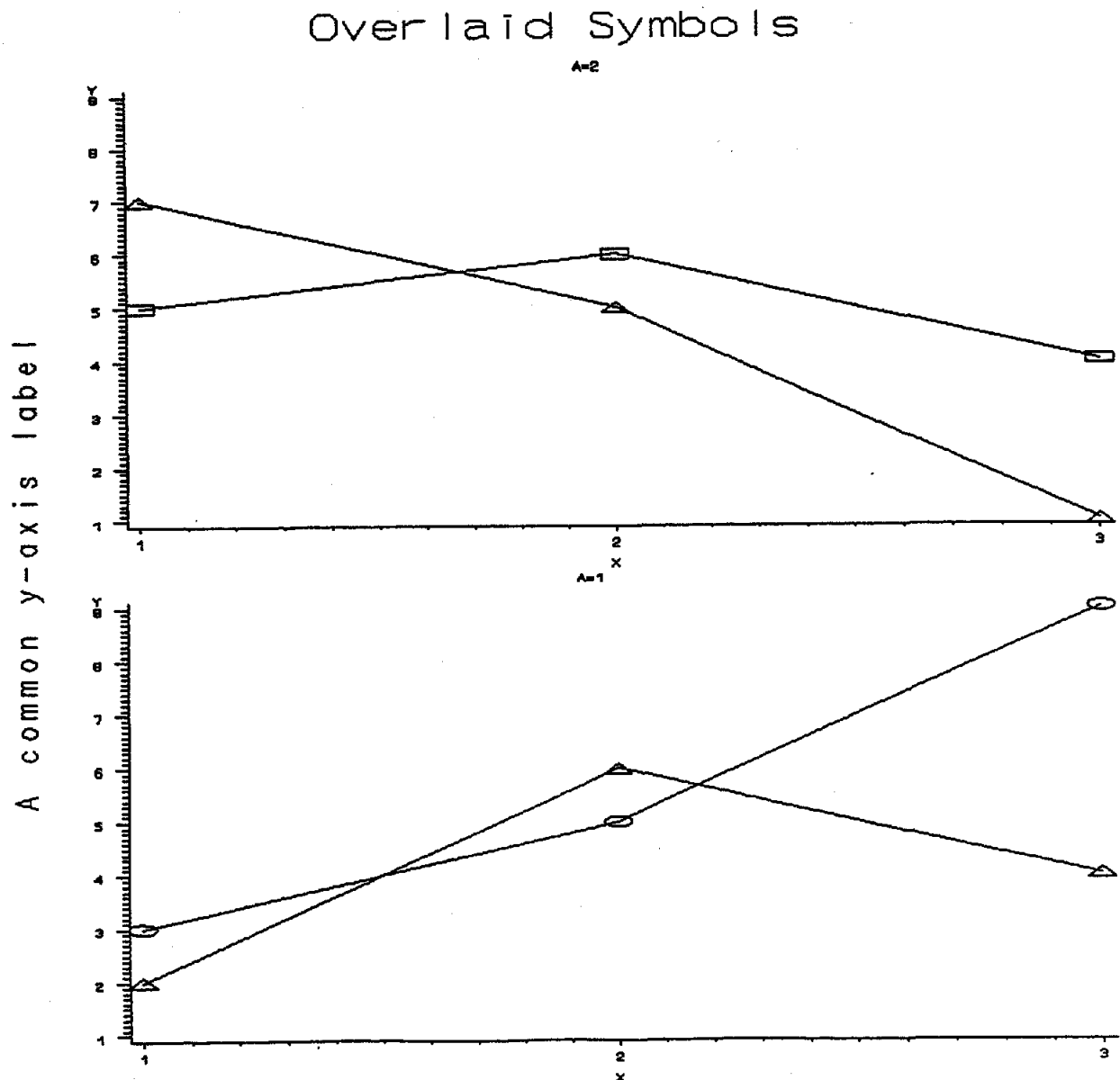


Figure 2. The template facility in PROC GREPLAY was used to simulate overlaid symbols.

Taking each observation in the A*B list, we get SAS to write out the necessary GPLOT code. The HAXIS= and VAXIS= PLOT options control the major ticks. The WHERE statement selects the observations for a given plot. We use the value of B to select the proper SYMBOL statement. A separate set of GPLOT statements are needed for each combination of A*B. All graphs are directed to a SAS graphics catalog using the GOUT= option. The SAS statements are written to the file "OUT1". After the statements are written to that file, we use the %inc "out1"; statement to get SAS to read that code and execute it.

Now, how to we get SAS to overlay all the graphs for a particular value of A onto a single page? PROC GREPLAY to the rescue again!!! We just need to create a template and place all the graphs for a particular value of A onto a single panel. We modify the code from the multi-graph example above to have three rectangular panels: panels one and two are equally sized to hold the graphs for the two values of A and yet leave room at the top of the page for a title and along the left edge for an axis label common to both graphs. The PROC GSLIDE code produces the titles and Y-axis label. Figure 2 is the final product.

ADDING FAKE DATA

Sometimes just adding a few observations to the desired dataset can greatly simplify the production of a graph. Keep this trick in mind whenever PATTERN or SYMBOL statements are used to control the appearance of the data area.

Let's suppose you wish to produce Figure 3. With one exception, it looks like an HBAR plot from GCHART. The difference is that the bars are not 'attached' to the vertical axis. Each bar is offset from the axis by a baseline value. The data represent some costs associated with a motor fleet for the second quarter of a year. For whatever reason, the boss doesn't want the miscellaneous charges drawn as a bar and yet wants them included in the total length of the bars. Thus the need to have the bars offset from the baseline. Table 1 lists the data to be charted.

Table 1. Sample data

<u>Cost Area</u>	<u>Month</u>	<u>Expenditure</u>
Repairs	April	4
Repairs	May	3
Repairs	June	5
Maintenance	April	2
Maintenance	May	4
Maintenance	June	5
Fuel	April	3
Fuel	May	5
Fuel	June	6

Well, we know that we need a PATTERN statement for each level (value) of the subgroup variable (month). The values for month are A (April), B (May), and C (June). We used a user-built format to associated the month names with these values.

In order to produce our chart, we added one additional month value for each group variable value (cost area). We gave the new level of month the value 'X' and mapped it within the user-written format to be blanks. Thus, our PROC FORMAT VALUE statement is now:

```
proc format;
value $month 'A'='April' 'B'='May' 'C'='June' 'X' = ' ';
```

Look at the order of the formatted values above - that will determine which levels of month go with which PATTERN statements. Since 'X' is mapped to the first alphabetical value (blank), it will be associated with the first PATTERN statement. We assign this PATTERN statement the same color as the CBACK= (background) color. Thus, the fill within the bar will not be visible to the reader. The remaining alphabetic order of the month values happens also to be the chronological order from April to June. The PATTERN statements for these values use the foreground color.

```
pattern1 v=empty c=white;
pattern2 v=x2 c=black;
pattern3 v=x3 c=black;
pattern4 c=x4 c=black;
```

The remaining problem we now have is the outline of the bars themselves. Just as with the first PATTERN statement, we set the COUTLINE= parameter on the HBAR statement to have the value of the background color. Now we have the appearance of floating bars. A complete listing of the code is found at the end of the paper.

Table 1. Sample data
Horizontal Bar Chart

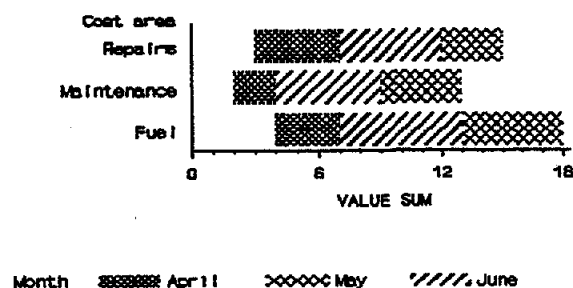


Figure 3. Floating horizontal bar chart

REPORT ENHANCEMENT

One can also turn to the graphical capabilities of SAS to solve other types of problems. SAS/GRAPH and the Data Step Graphics Interface (DSGI) allow users much greater control over output devices. Thus, embellishments and finer detail may be added to standard SAS output. The background shading in Table 2 was added using the DSGI. Code for this example may be found at the end of the paper.

The steps to follow are simple. First, instead of directing your output to either the printer or the Display Manager OUTPUT window, send it to a disk file. For SAS procedures, you can do this step with PROC PRINTTO. For reports written by a Data Step (as in our example), just use a FILE statement.

Since our report is now on disk, we can treat it as an input file. We read the report line-by-line. The LRECL= and PAD options on the INFILE statement allow each line to be treated as if it were the same length. At the beginning of the file, we initialize the DSGI and then set up our font and pattern fills. This is particularly effective on PostScript printers. The key, though, is to realize that SAS starts each page with a form feed (ASCII character 12 on most platforms). This allows the user to establish an anchor for each page. Any line on that page can be addressed if the

number of lines since the form feed can be determined. In our example, we know that we need to shade every odd line after line number 4. We write out each line until we find the next form feed or end-of-file character.

SUMMARY

These are just a few of the tricks that veteran SAS programmers use to enhance and simplify their SAS work. If you need to combine graphs, you can use the TEMPLATE facility in PROC GREPLAY. Remember that you can also use GREPLAY to produce hardcopies of graphs that have been enhanced with the Graphics Editor.

Anytime that a discrete variable is associated with PATTERN or SYMBOL statements, you can add a few extra observations to help control the appearance of the data area or axes. You can hide these extra observations by setting the appropriate color parameters to the graphic background color.

The final trick involves the Data Step Graphics Interface. As opposed to the SAS/GRAPH procedures which work on the columns of a SAS data set, the DSGI operates on the observations within a data set. Thus, the DSGI is useful anytime you are working on a row-by-row basis.

•SAS and SAS/GRAPH are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. • indicates USA registration.

Sample Program 1

This program pertains to Figure 2. When you execute the program, the code written by the last DATA _NULL_ step is an example of PROC GREPLAY template described in the text for Figure 1 ("Text on Multi-graphs").

- * produce two graphs - one for each value of A;
- * the graphs are simple scatterplots of the form $Y \sim X=B$
- * where B specifies which symbol statement to use. the
- * problem is to get both graphs for a given A onto a
- * single page;

- * read in some simple data;

```
data test;
input a b x y;
cards;
1 1 1 3
1 1 2 5
1 1 3 9
1 3 1 2
1 3 2 6
1 3 3 4
2 2 1 5
2 2 2 6
2 2 3 4
2 3 1 7
2 3 2 5
2 3 3 1
;
```

- * set up the desired SYMBOL statements. do not forget
- * the C= parameter or else SAS/GRAPH will just use
- * SYMBOL1 for each value in the COLORS= list for your
- * device!!;

```
symbol1 v=circle height=2 i=join c=yellow;
symbol2 v=square height=2 i=join c=yellow;
symbol3 v=triangle height=2 i=join c=yellow;
FOOTNOTE;
title;
```

- * get a list of the B values within each value of A;

```
proc freq data=test;
tables a*b / out=freq1(drop=count percent) noprint;
run;
```

- * find the number of B values for each value of A;

```
proc freq data=freq1;
tables a / out=freq2(drop=percent) noprint;
run;
```

- * get the extreme values of X and Y so we can control
- * the axes;

```
proc means data=test noprint;
var x y;
output out=varstats min=minx miny max=maxx maxy;
run;
```

```
goptions device=cgmmp ftext=hwcgm005 nodisplay;
```

- * now for the each observation in the A*B list, get SAS
- * to write out the necessary GLOT code to produce that
- * specific graph. we use the haxis and vaxis PLOT
- * options to control the axes. we use the WHERE
- * statement to select the observations for a given plot.
- * we use the value of B to select the proper SYMBOL
- * statement. we save the graphs using the GOUT= option
- * so we can replay them later;

```
data _null_;
set freq1 end=eof;
if _n_=1 then set varstats;
```

```
retain minx miny maxx maxy;
file "out1";
put "proc gplot data=test gout=saver;" /
"title1;" / "by a;" /
"plot y*x=" b " / haxis=" minx " to " maxx
" vaxis=" miny " to " maxy ";" / "where a=" a
" & b=" b ";" /
"run;";
if eof then put "quit;";
run;
```

- * execute the GLOT code just written;

```
%inc "out1";
proc gslide gout=saver;
title1 h=1.5 "Overlaid Symbols";
note move=(-30pct, -65pct) la=90
"A common y-axis label";
footnote1 h=1
"Figure 2. The template facility in PROC GREPLAY was";
footnote2 h=1
" was used to simulate overlaid symbols. ";
run;
```

```
goptions display gsfname=grafout gsfmode=replace;
filename grafout "a:\marsh\art2.gsf";
```

- * now get SAS to write some GREPLAY code to place all
- * graphs for a given value of A onto a single page. we
- * set up a template with 1 panel that covers the whole
- * page. then we pull all graphs for a given A from the
- * catalog created by GLOT and place those graphs onto
- * that 1 panel;

```
data _null_;
set freq2 end=eof;
file "out2";
retain base 0 panel 1;
if _n_=1 then
put "proc greplay igout=saver nofs;" /
"tc overlaid;" /
"tdef v2t 1/ llx=5 lly=5 lrx=100 lry=5
urx=100 ury=50 ulx=5 uly=50"
/ " 2/ copy=1 xlatey=45"
/ " 3/ llx=0 lly=0 lrx=100
lry=0 urx=100 ury=100 ulx=0 uly=100;" /
"template v2t;" /
"treplay ";
do i=1 to count;
y = base+i;
put panel ":" y;
end;
base+count; panel+1;
if eof then put "3:5; run; quit;";
run;
```

```
%inc "out2";
```

Table 2.
Using the Data Step Graphics Interface to Enhance a Report

38	48	58	68	78	88	98	108	118
39	49	59	69	79	89	99	109	119
40	50	60	70	80	90	100	110	120
41	51	61	71	81	91	101	111	121
42	52	62	72	82	92	102	112	122
43	53	63	73	83	93	103	113	123
44	54	64	74	84	94	104	114	124
45	55	65	75	85	95	105	115	125
46	56	66	76	86	96	106	116	126
47	57	67	77	87	97	107	117	127
48	58	68	78	88	98	108	118	128
49	59	69	79	89	99	109	119	129
50	60	70	80	90	100	110	120	130
51	61	71	81	91	101	111	121	131
52	62	72	82	92	102	112	122	132
53	63	73	83	93	103	113	123	133
54	64	74	84	94	104	114	124	134
55	65	75	85	95	105	115	125	135
56	66	76	86	96	106	116	126	136
57	67	77	87	97	107	117	127	137
58	68	78	88	98	108	118	128	138
59	69	79	89	99	109	119	129	139
60	70	80	90	100	110	120	130	140

Sample Program 2

The following program produced the floating horizontal bar chart in Figure 3.

```
proc format;
value group 1='Repairs' 2='Maintenance' 3='Fuel';
value $class 'A'='April' 'B'='May' 'C'='June' 'X' =
' ';
run;

data float;
input group class $ value;
format group group. class $class.;
label group="Cost area" class="Month";
cards;
1 X 3
2 X 2
3 X 4
1 A 4
1 B 3
1 C 5
2 A 2
2 B 4
2 C 5
3 A 3
3 B 5
3 C 6
;
run;

options ls=55;
proc print;
title "Table 1. Sample data";
title2 "Horizontal Bar Chart";
where class^="X";
run;

goptions device=cgmwp ftext=hwcgm005 cback=white
colors=(black)
gsfname=grafout gsfmode=replace;
filename grafout "a:\marsh\art3.gsf";
pattern1 v=empty c=white;
pattern2 v=x2 c=black;
pattern3 v=r3 c=black;
pattern4 c=x4 c=black;

proc gchart;
title h=1.5 "Table 1. Sample data";
title2 h=1.5 "Horizontal Bar Chart";
hbar group / discrete sumvar=value subgroup=class
outline=white raxis=0 to 18 by 6 nostats;
run;
quit;
```

Sample Program 3

This code was used to generate some dummy data and then produce Table 2.

```
title;
footnote;

* simulate some data. we assume that K1-K10 are ten
* items to printed on a report. in practice, you would
* do all data steps and/or
* procs that compute the data to be printed;
data a;
array k {10} k1-k10;
do j=1 to 60;
do i=1 to 10;
k(i) = (i-1)*10+j;
end;
output;
end;
keep k1-k10;
run;

* now, you will "print" the data to disk. you do this
* step to take advantage of the ability of SAS to
* produce page headings and to control any BY-group
* processing that is needed;
filename trick1 "tricky.prn";
options ls=130 ps=44;
data _null_; set a;
array k {10} k1-k10;
file trick1 print header=newpage;
do i=1 to 10;
m = (i-1)*10+1;
put am k(i) a;
end;
put;
return;
newpage: put / "Table 2.";
put "Using the Data Step Graphics Interface to Enhance
a Report" ///; return;
run;

* now read back in the "printed" report and send it to
* the hardcopy postscript device. you can use the
* GOPTIONS statement to control the page orientation and
* type of hardcopy device;
goptions rotate=landscape noprompt;

data _null_;

* create the variables to read each line (A) and to
* detect form feeds within the "printed" report (FF);
length a $ 130 ff $ 1;
retain lineno; ff = byte(12);

* file containing the "printed" report;
infile trick1 lrecl=130 pad end=eof;

* initialize the SAS data step graphics interface. to
* be able to shade every other line in the report, we
* will treat the reproduction of the report as if it
* were a graph;
if _n_=1 then do;
rc = ginit();
input a;
* discard the initial formfeed at beginning of report;
end;

* clear the graphic window, set the color map, fill
* types, and print area for the new page;
rc = graph('clear');
```

GRAPHICS

```
rc = gset('colrep', 1, 'black');
rc = gset('colrep', 2, 'gray33');
rc = gset('colrep', 3, 'gray88');
rc = gset('colrep', 4, 'graycc');

rc = gset('filtype', 'solid');
rc = gset('filcolor', 4);
rc = gset('texcolor', 1);
rc = gset('texfont', 'hwp1001');
* note - you may have to played with the height of the
* text to handle an entire line of text;
rc = gset('texheight', 2.00);

* read the lines of the report until formfeed found;
ff_found=0;
lineno=0;
* keep track of the # of lines on current page;
do until(ff_found=1 or eof);
  input a 1-130;
  lineno=lineno+1;

  * does the line have a form feed in it?;
  if index(a, ff)>0 then ff_found=1;

  * do this for lines that are more than just the
  * formfeed;
  *
  * note: for the SAS PostScript device, it assumes
  * 100 units vertically and 133 units horizontally;
  if ff_found=0 then do;
    * shade every other line in table part of each
    * page; 2.25 is current text height. .45 is used
    * to center the gray background around each even
    * line number text;
    if lineno>4 and mod(lineno, 2) = 1 then
      rc = gdraw('bar', 0, 100-lineno*2.25-.45, 133,
100-(lineno+1)*2.25-.45);

    * add input line to graph page;
    rc = gdraw('text', 0.00, 100-lineno*2.25, a);
  end;
end;

* end of file is found - terminate the work;
if eof then do; rc = gterm(); stop; end;

* whole page is ready - produce it;
rc = graph('update');

run;
```