%CYARROW, A NEW ANNOTATE MACRO

Chang Y. Chung, Princeton University

Ya Huang, Amylin Pharmaceuticals, Inc.

ABSTRACT

We present a macro, %cyArrow, which makes it easy to draw arrows using SAS®/GRAPH's superior annotate facility. %cyArrow is specifically designed to be used in conjunction with %annomac macros, a set of SAS macros that provides a shortcut when creating an annotate dataset. %cyArrow is an alternative to the undocumented (and "broken") %arrow macro. In addition to honoring the five standard options for the 'draw' annotate function (color, hsys, line, size, and when), %cyArrow offers sophisticated arrow shape options that control the length and the angle of the arrow tip and its aspect ratio.

ANNOTATE DATA SET AND ANNOTATE MACROS

The annotate facility is a part of SAS/GRAPH. It is most commonly used to enhance graphics output by adding text and other graphic elements on them. It also creates advanced, custom, graphics output from scratch in conjunction with proc ganno.

In order to use the annotate facility, one creates a dataset (an annotate dataset) with specific variables and values. The annotate dataset then is fed into a graphics procedure. The procedure, in turn, processes each observation of the annotate dataset as an instruction to the facility. A good introduction to the annotate facility is the, delightful, short volume by Carpenter(1999a).

Since the bulk of work in using annotate facility lies in creating the annotate dataset, SAS kindly provides a set of macros, %annomac, which significantly reduces the amount of work. As Rorie and Duncan(2003) demonstrate (also see Carpenter(1999b)), a single, %line( ) annotate macro call can replace: (1) creating two observations; (2) assigning the function variable values ("move", for the first observation; and "draw" for the second); (3) assigning x and y coordinates for the starting and ending points of a line; (4) assigning color, line type, and thickness.

The %annomac includes %bar to create a fill-able rectangle, %line to draw a line, %slice to draw a pie slice, %label to write text at the specified location, to name a few. Our installation of SAS Release 9.1.2 includes the source code file for the %annomac macros (in the "SAS 9.1/core/sasmacro" directory). To our delight, the source code includes an undocumented macro called %arrow.

BROKEN %ARROW

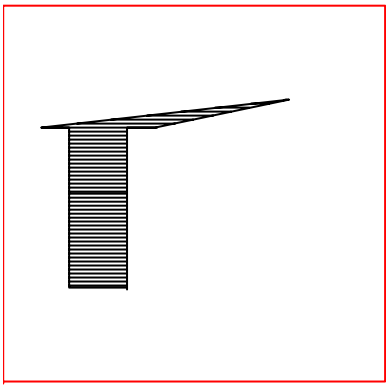
When we try to use the %arrow macro, however, we come to an understanding of the reason why it is not documented -- it is broken. Disappointed, but motivated, we end up writing our own annotate macro, %cyArrow.

```
Filename gout “%sysfunc(pathname(WORK))/broken.emf";

goptions reset=all
  device=emf gsfname=gout gsfmode=replace
  hpos=40 vpos=40 hsize=2in vsize=2in
;

data anno;
  %annomac(nomsg)
  %dclanno
  %frame(CXFF0000, 1, 1)
  %arrow(10,10,30,30,6,black,1)
run;

proc ganno annotate=anno;
run;
quit;
```



%CYARROW

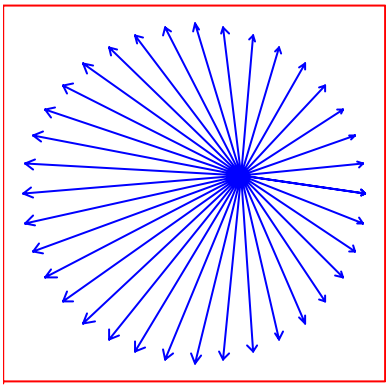
The new macro draws a very simple arrow, but does it well. Here is an example.

```
filename gout "%sysfunc(pathname(WORK))/cy1.emf";

goptions reset=all
  device=emf gsfname=gout gsfmode=replace
  hpos=40 vpos=40 hsize=2in vsize=2in
;

data anno;
  %annomac(nomsg)
  %dclanno
  %frame(CXFF0000, 1, 1)
  radius = 16;
  do deg = 0 to 360 by 10;
    rad = deg * constant('pi') / 180;
    cx = 20 + radius * cos(rad);
    cy = 20 + radius * sin(rad);
    %cyArrow(25,22,cx,cy,color="cx0000ff")
  end;
run;

proc ganno annotate=anno;
run;
quit;
```



HOW TO USE

`%cyArrow` is designed to be used with the `%annomac` macros. The macro follows the `%annomac` convention in macro implementation and usage. For instance, it assumes that users issue `%dclanno` macro to set up required variables, just as other annotate macros do.

Implementation wise, `%cyArrow` features much improved internals: it has robust input parameter checking and error handling routines; it minimizes unwanted side-effects using stacks. Still, it is very simple to use.

For example, the following short line draws an arrow.

```
%cyArrow(1,1,3,3) /* draw an arrow starting from (x,y)=(1,1), pointing to (3,3) */
```

There are two sets of options. The first group of five options (`color`, `line`, `size`, `hsys`, and `when`) are the same as the options for the "`draw`" and other annotate functions.

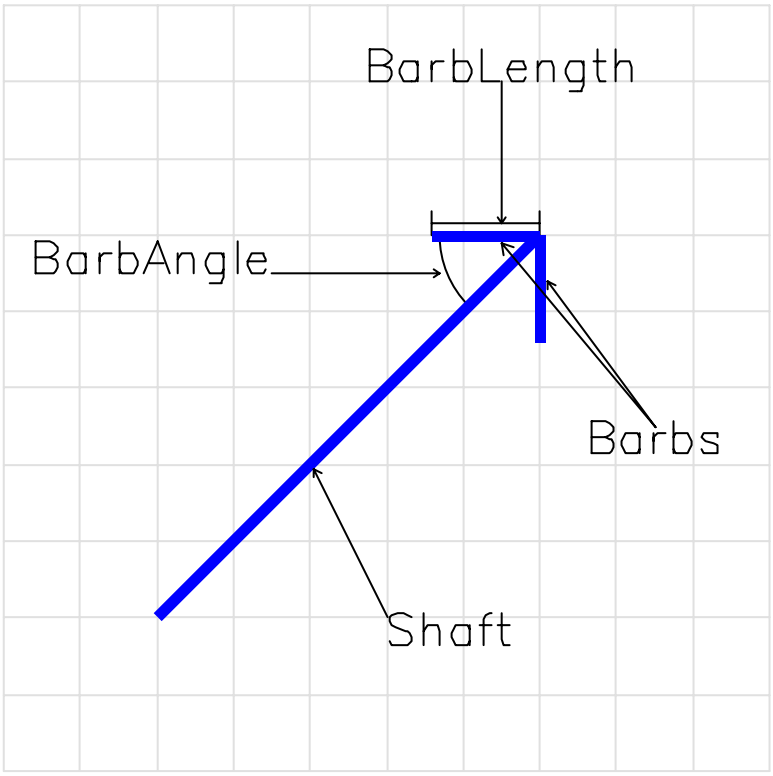
`color` controls the color of the arrow. It can take a string literal or a character variable of length $8.

```
%cyArrow(x1, y1, x2, y2, color = "cx0000ff") /* blue */
```

`line` specifies the line type – values range from 0 (no line) to 46 (three dots and dashes). `size` controls the thickness of the line, the unit of which depends on `hsys`. For example, when `hsys="4"`, then `line=1` indicates that the arrow lines should be one cell unit thick. `%cyArrow` inherits the `hsys` value that is assigned before the `%cyArrow` call (presumably using `%declanno` or `%system` annotate macros). But you can temporarily override the unit with `hsys=` option.

`when=` option controls whether the arrow should be drawn before ("`B`") or after ("`A`") the graphs, when the annotate dataset is used with SAS/GRAPH procedures like `proc gplot`.

The second group of options control how the tip of the arrow should be drawn. An arrow has a "shaft", and two "barbs," as shown below.




```
%include "cyArrow.sas";

data anno;
  %annomac
  %dclanno
  length text $20;
  %system(4,4,4)

  /* grid */
  drop i;
```

```
    do i = 0 to 100 by 10;

        %line(i,0, i,100, cxe0e0e0, 1, 1)
        %line(0,i, 100,i, cxe0e0e0, 1, 1)
    end;

    /* the large arrow */
    %cyArrow(20, 20, 70, 70, color="cx0000ff", size=5, barbLength=0.2, barbAngle=45)

    /* an arrow pointing "shaft" */
    %cyArrow(50, 20, 40.5, 39.5, color="cx000000", size=0.05)
    %label(  50, 20, "Shaft", cx000000, 0, 0, 7, Simplex, 6)

    /* arrows pointing "barbs" */
    %cyArrow(85, 45, 65, 69, color="cx000000", size=0.05)
    %cyArrow(85, 45, 71, 64, color="cx000000", size=0.05)
    %label(  85, 45, "Barbs", cx000000, 0, 0, 7, Simplex, 5)

    /* barbLength */
    drop j;
    j = 70 - sqrt(2) * 10;
    %line(j,  71.5, 70, 71.5, cx000000, 1, 0.01)
    %line(j,  70,    j, 73,   cx000000, 1, 0.01)
    %line(70, 70,   70, 73,   cx000000, 1, 0.01)
    %cyArrow(65, 90, 65, 71.5, color="cx000000", size=0.05)
    %label(  65, 90, "BarbLength", cx000000, 0, 0, 7, Simplex, B)


    /* barbAngle */
    when = "A";
    %slice(  70, 70, 180, 45, 13, cx000000, empty, 0)
    %cyArrow(35, 65, 57,  65, color="cx000000", size=0.05)
    %label(  35, 65, "BarbAngle", cx000000, 0, 0, 7, Simplex, A)
  run;

  dm 'graph1; end;' wedit;
  filename gout "shaftAndBarb.emf";
  goptions reset=all vsize=4in hsize=4in vpos=100 hpos=100
    device=emf gsfname=gout gsfmode=replace
    targetdevice=emf goutmode=replace
  ;

  proc ganno annotate=anno;
  run;
  /*
  data one;
    retain one_y one_x .;
  run;
  proc gplot data=one;
    plot one_y * one_x
       / haxis=(0 to 1 by 1) vaxis=(0 to 1 by 1) annotate=anno;
  run;
  quit;
  */
  goptions reset=all;
```

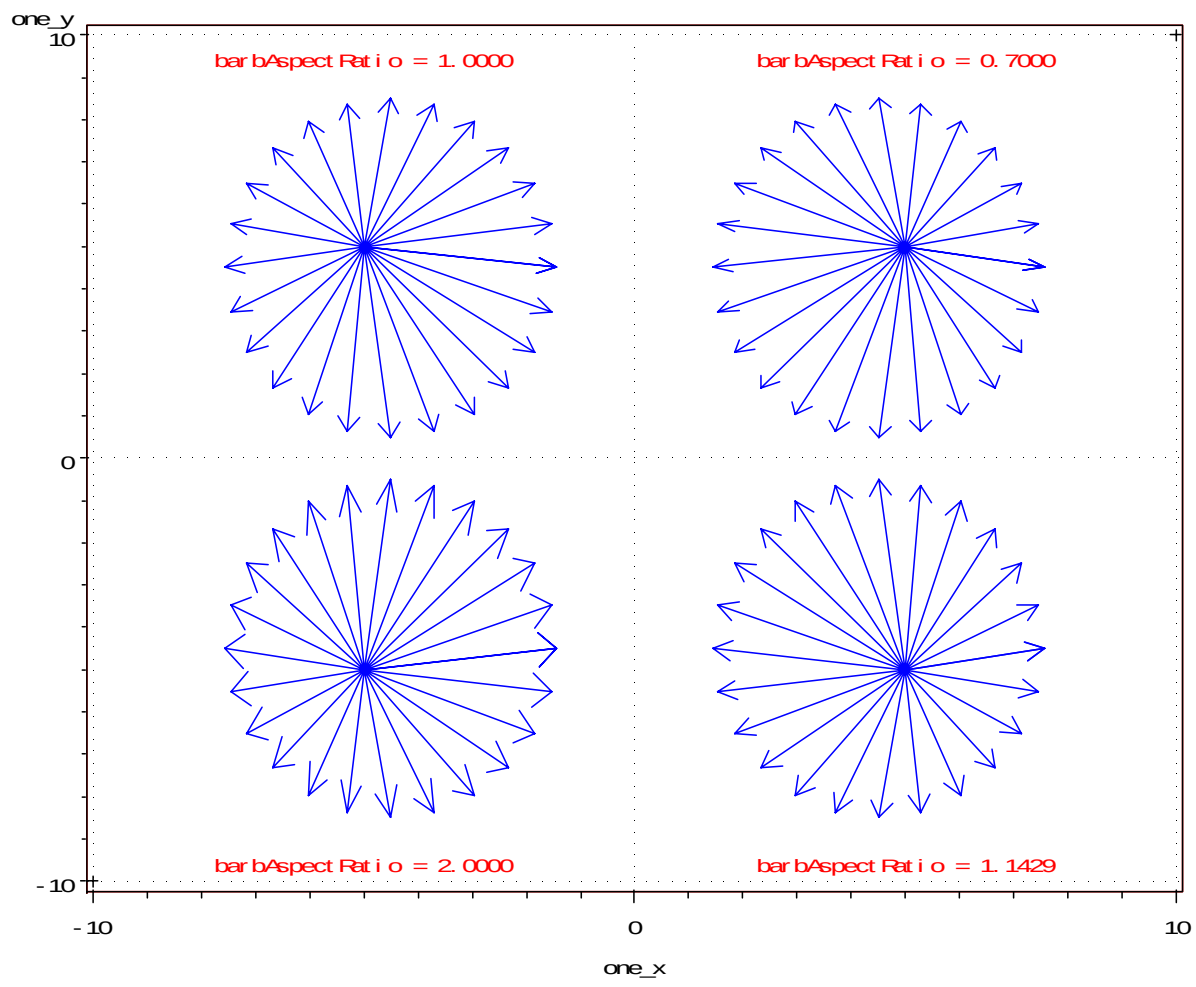
`barbAngle` option allows users to change the angle (in degrees between 1 and 90) between a barb and the shaft. The default value is 30.

The length of the barbs is controlled by two options: `barbLengthType` and `barbLength`. The `barbLengthType` can be either "`F`" (for Fixed) or "`P`" (for Proportional). The default is "`P`," under which, the value specified in `barbLength` indicates length as a proportion of the shaft's. The default value is 0.05, meaning

5 percent of the shaft length. When the `barbLengthType="F"`, on the other hand, the `barbLength` indicates the absolute length in `hsys` unit.

An aspect ratio refers to the ratio of the width(x) to height(y). Since the shaft is already defined by the two end-points, only the barbs are affected by the aspect ratio. The barbs' length and angle are adjusted by `barbAspectRatio`.

The following shows the effects of the `barbAspectRatio`. The aspect ratio of the graphic area is 1.1429, since we specify `HSize = 8 in` and `VSize = 7in` (8 / 7 = 1.1429) in the `goptions` statement. With the default (`barbAspectRatio=1`), the barbs are not of equal length and the angles between a barb and the shaft are different. And the amount of "distortion" depends on the length of the arrow and the direction. The same kind of distortion happens when the `barbAspectRatio` is equal to 0.7 or 2.0. By specifying the `barbAspectRatio` which is the same as that of the graphic area (here 1.1429), the arrows look best.




```
%include "cyArrow.sas";

/*-- test data set ------------------------------------------------*/
data one;
  one_x =-10; one_y =-10; output;
  one_x = 10; one_y = 10; output;
run;

data anno;
```

```
  %annomac(nomsg)
  %dclanno
  %system(2,2,2)
  %frame(CXFF0000, 1, 1)

  length text $30;
  radius = 3.5;

  %macro doCircle(ar=, labelX=, labelY=
    ,offsetX=, offsetY=, centerX=, centerY=
  );
    function="label";
    text="barbAspectRatio = &ar.";
    font="simplex";
    x=&labelX.;
    y=&labelY.;
    size=0.5;
    output;
    do deg = 0 to 360 by 15;
      rad = deg * constant('pi') / 180;
      xx =  &offsetX. + radius * (1/1.1429) * cos(rad);
      yy =  &offsetY. + radius * (1.1429)   * sin(rad);
      %cyArrow(&CenterX., &CenterY., xx, yy
        , barbLengthType="Fixed"
        , barbLength=0.4, barbAspectRatio=&ar.
        , color="cx0000ff", size=0.5
      )
    end;
  %mend doCircle;
  %doCircle(ar=1.0000, labelX=-5, labelY=9.5
    , offsetX=-4.5, offsetY= 4.5, centerX=-5, centerY= 5)
  %doCircle(ar=0.7000, labelX= 5, labelY=9.5
    , offsetX= 4.5, offsetY= 4.5, centerX= 5, centerY= 5)
  %doCircle(ar=1.1429, labelX= 5, labelY=-9.5
    , offsetX= 4.5, offsetY=-4.5, centerX= 5, centerY=-5)
  %doCircle(ar=2.0000, labelX=-5, labelY=-9.5
    , offsetX=-4.5, offsetY=-4.5, centerX=-5, centerY=-5)
run;

dm log 'graph1; clear; end;' wedit;   /* close the graph1 window */
goptions goutmode=replace; /* entire contents of catalog replaced */

/*-- aspect ratio test - gplots---------------------------------*/
%macro doGPlot(hsize=, vsize=, labelSize=);
  dm 'graph1; end;' wedit;
  filename gout "shaftAndBarb.emf";
  goptions reset=all hsize=&hsize. vsize=&vsize.
    device=emf gsfname=gout gsfmode=replace
    targetdevice=emf goutmode=replace
  ;
  proc gplot data=one;
    title font="Tahoma" "HSize=&hsize. and VSize=&vsize.";
    plot one_y*one_x/ annotate=anno grid;
  run;
  quit;
  title;
  goptions reset=all;
%mend doGPlot;
%doGPlot(hsize=8 in, vsize=7 in)
```

HOW IT WORKS

The source code for the %cyArrow is presented below. The code is commented and formatted to enhance readability.

Early on, we made some design choices. Recognizing that this is a "data-step macro" (which is to be called within a data step) we decide to minimize unwanted side-effects that might cause problems like variable and label name collisions. Thus we prefix all the names (including data step label, macro label, variables, and macro variables) by utilizing a random name generated for each %cyArrow macro call (see line marked with (1)). At the end of the macro, all the temporary variables are listed in the drop statement (7).

%cyArrow checks input errors at both macro compile time and data step run time. The macro compile time error checks are simply to see if any parameter is empty. Most of the error checks are done at data step run time – the data step code for error handling is simple enough that we implements it using a simple parameterized string substitution in lieu of subroutine calls (See (2) and (3)).

Unwanted side effects may be caused by "draw" function, since it leaves behind the last coordinates in the x[y]last variables: xlast, ylast, xlastt, and ylastt. Since we require both the start and end point coordinates in %cyArrow, there is no need for keeping the last coordinates. So we decide to preserve the values of these variables same as before the %cyArrow is called -- by storing the current x[y]last values before using "draw" function and restore them when we are done with drawing arrows. The "push" and "pop" annotate functions work very well in this context ((4) and (6)).

In order to ease the maintenance burden, we decide not to call other annotate macros inside %cyArrow. This also enables us to write compact data step code (5).

```
%macro cyArrow(
                          /* coordinates --------------------------- */

    x1, y1                /* starting point coordnate -- required    */
  , x2, y2                /* end point coordinate     -- required    */


                          /* cyArrow specific options -------------- */

  , barbAngle      =30    /* angle between barb and shaft in degrees */
  , barbLengthType ="P"   /* [F]ixed or [P]roportional to shaft len  */
  , barbLength     =0.05  /* if type=F then absolute length in hsys  */
                          /*   unit. otherwise proportion of shaft   */
                          /*   length                                */
  , barbAspectRatio=1     /* the ratio of width(x) to height(y)      */


                          /* draw function options ----------------- */

  , color          ="*"   /* color codes.                            */
  , hsys           =hsys  /* coord sys for size option               */
  , line           =1     /* line type 1...46                        */
  , size           =0.5   /* thickness of lines in hsys units        */
  , when           ="A"   /* annotate "A"fter gproc outputs          */

);

%*-- draw a simple arrow using annotate facility. This replaces     --*;
%*--   the undocumented macro %arrow()                               --*;
%*-- by chang y chung and ya huang                                   --*;
%*-- v1.0 on 2004-07-26                                             --*;
```

```
%*-- v1.5 on 2004-07-26 fixed the exit part                          --*;


%*-- helpers ------------------------------------------------------*;

     %*-- a random prefix for "global" things ----------------------*;
     %global cyArrow;
     %let cyArrow = %sysfunc(putn(%sysfunc(int(1e8*                                  (1)
          %sysfunc(ranuni(0)))),z8.));

     %*-- for data step runtime error handling ----------------------*;
     %local err cond msg;                                                            (2)
     %let err = %nrstr(
          if %unquote(&cond.) then do;
               put "&preMsg. %unquote(&msg.). &postMsg.";
               function = "comment";
               output;
               goto _&cyArrow._exit;
          end;
     );
     %local preMsg postMsg commentAndExit;
     %let preMsg = NOTE: (cyArrow);
     %let postMsg= arrow not drawn.;



%*-- macro compile time error check ---------------------------------*;

     %*-- no empty parameters. --------------------------------------*;
     %local params param value i;
     %let params = x1 y1 x2 y2 barbAngle barbLength barbLengthType
          barbAspectRatio color hsys line size when;
     %let i = 1;
     %let param = %scan(&params.,&i);
     %do %while (&param.^=);
          %if %superq(&param.)= %then %do;
               %put &preMsg. &param. should not be blank. &postMsg.;
               %goto exit;
          %end;
          %let i = %eval(&i. + 1);
          %let param = %scan(&params., &i.);
     %end;

%*-- data step runtime check ----------------------------------------*;

     %*-- coordinates cannot be missing ----------------------------*;
     %local i coord coords;
     %let coords = x1 y1 x2 y2;
     %do i = 1 %to 4;
          %let coord = %scan(&coords., &i.);
          _&cyArrow._&coord. = %unquote(&&&coord..);
          %let cond = missing(_&cyArrow._&coord.);
          %let msg  = &coord. should not be missing;                                 (3)
          %unquote(&err.)
     %end;

     %*-- barbs ----------------------------------------------------*;
     _&cyArrow._barbAngle = (%unquote(&barbAngle.));
     %let cond = not (0<=_&cyArrow._barbAngle<=90);
     %let msg  = barbAngle should be between 0 to 90 degrees;
     %unquote(&err.)

     _&cyArrow._barbLengthType = upcase(trim(left(
```

```
          %unquote(&barbLengthType.)));
     %let cond = missing(_&cyArrow._barbLengthType);
     %let msg  = barbLengthType should not be missing;
     %unquote(&err.)

     _&cyArrow._barbLengthType = substr(_&cyArrow._barbLengthType,1,1);
     %let cond = not (_&cyArrow._barbLengthType in ("P" "F"));
     %let msg  = barbLengthType should be either [P]roportional
          or [F]ixed;
     %unquote(&err.)

     _&cyArrow._barbLength = (%unquote(&barbLength.));
     %let cond = not (0 <= _&cyArrow._barbLength);
     %let msg  = barbLength should not be negative;
     %unquote(&err.)

     if _&cyArrow._barbLengthType = "P" then do;
          %let cond = not (_&cyArrow._barbLength<=1.0);
          %let msg  = Proportional type barbLength should be/*
               */ between 0 and 1;
          %unquote(&err.)
     end;

     _&cyArrow._barbAspectRatio = (%unquote(&barbAspectRatio.));
     %let cond = not (0 < _&cyArrow._barbAspectRatio);
     %let msg  = barbAspectRatio should be larger than zero;
     %unquote(&err.)


%*-- calculation for the shaft ---------------------------------------*;

     %*-- always adjust y for aspect --------------------------------*;
     _&cyArrow._ay1 = _&cyArrow._y1 * _&cyArrow._barbAspectRatio**-1;
     _&cyArrow._ay2 = _&cyArrow._y2 * _&cyArrow._barbAspectRatio**-1;

     %*-- calculate shaft angle and length -------------------------*;
     _&cyArrow._shaftLength = sqrt(
            (_&cyArrow._x1  - _&cyArrow._x2)**2
          + (_&cyArrow._ay1 - _&cyArrow._ay2)**2
     );
     %*-- check ----------------------------------------------------*;
     %let cond = _&cyArrow._shaftLength <= 0;
     %let msg  = shaft length <= 0;
     %unquote(&err.)

     %*-- direction ------------------------------------------------*;
     _&cyArrow._shaftDirection = atan2(
            _&cyArrow._ay1 - _&cyArrow._ay2
          , _&cyArrow._x1  - _&cyArrow._x2
     );

%*-- calculation for the barbs ---------------------------------------*;

     %*-- angle ----------------------------------------------------*;
     _&cyArrow._barbAngle  =
          _&cyArrow._barbAngle * constant('pi') / 180
     ;
     if _&cyArrow._barbLengthType = "P" then do;
          _&cyArrow._barbLength =
               _&cyArrow._shaftLength * _&cyArrow._barbLength
          ;
     end;
     %*-- check ----------------------------------------------------*;
```

```
     %let cond = _&cyArrow._barbLength <= 0;
     %let msg  = barb length <= 0;
     %unquote(&err.)


     %*-- coordinates ------------------------------------------------*;
     _&cyArrow._barbX1 = _&cyArrow._x2 + _&cyArrow._barbLength
          * cos(_&cyArrow._shaftDirection + _&cyArrow._barbAngle);
     _&cyArrow._barbY1 = _&cyArrow._y2 + _&cyArrow._barbLength
          * sin(_&cyArrow._shaftDirection + _&cyArrow._barbAngle)
          * _&cyArrow._barbAspectRatio;
     _&cyArrow._barbX2 = _&cyArrow._x2 + _&cyArrow._barbLength
          * cos(_&cyArrow._shaftDirection - _&cyArrow._barbAngle);
     _&cyArrow._barbY2 = _&cyArrow._y2 + _&cyArrow._barbLength
          * sin(_&cyArrow._shaftDirection - _&cyArrow._barbAngle)
          * (_&cyArrow._barbAspectRatio);


%*-- save xlast, ylast, xlastt, ylastt and other options--------------*;
     function = "push";                                                  (4)
     output;
     _&cyArrow._color = trim(color);
     _&cyArrow._line  = 1 * line;
     _&cyArrow._size  = 1 * size;
     _&cyArrow._hsys  = trim(hsys);
     _&cyArrow._when  = trim(when);


%*-- common vars to draw function -----------------------------------*;
     color = %unquote(&color.); /* codes only */
     hsys  = %unquote(&hsys.);  /* for size      */
     line  = %unquote(&line.);  /* 1, 2, ..., 46 */
     size  = %unquote(&size.);  /* in hsys unit  */
     when  = %unquote(&when.);  /* draw before/after the proc output */


%*-- draw "shaft" ----------------------------------------------------*;
     function = "move";                                                  (5)
          x = _&cyArrow._x1;
          y = _&cyArrow._y1;
     output;
     function = "draw";
          x     = _&cyArrow._x2;
          y     = _&cyArrow._y2;
     output;


%*-- draw "barbs" ----------------------------------------------------*;
     %do i = 1 %to 2;
         function = "move";
             x = _&cyArrow._barbX&i.;
             y = _&cyArrow._barbY&i.;
         output;
         function = "draw";
             x     = _&cyArrow._x2;
             y     = _&cyArrow._y2;
         output;
     %end;


%*-- restore saved values -------------------------------------------*;  (6)
     function = "pop";
     output;
```

```
      color = _&cyArrow._color;
      line  = _&cyArrow._line;
      size  = _&cyArrow._size;
      hsys  = _&cyArrow._hsys;
      when  = _&cyArrow._when;


  %*-- exits ------------------------------------------------------------*;
       %exit:;                                                              (7)
       _&cyArrow._exit:;
            drop _&cyArrow._:;


  %mend  cyArrow;
```

SUMMARY

We have presented a macro, `%cyArrow`, to be used with `%annoMac` in SAS/GRAPH, along with several examples. The `%cyArrow` draws simple, but correct arrows. `%cyArrow` honors most of the options that work with the 'draw' annotate function (`color`, `hsys`, `line`, `size`, and `when`). It also has sophisticated arrow shape control options such as the length and angle of the arrow tip and its aspect ratio.

REFERENCES

Carpenter, Art (1999a). *Annotate: Simply the Basics.* BBU Press (SAS). ISBN: 1-58025-578-7. A companion web page at: http://www.sas.com/apps/pubscat/bookdetails.jsp?pc=57320.

Carpenter, Art(1999b). "Using ANNOTATE MACROS as Shortcuts." Information Visualization section paper. Paper 168. *SUGI 24 Proceedings.* Archived at http://www2.sas.com/proceedings/sugi24/Infovis/p168-24.pdf.

Dias, Carlos Tadeu Santos (2004). "arrows." A SAS-L posting on Jun. 23, 2004. Archived at http://listserv.uga.edu/cgi-bin/wa?A2=ind0406D&L=sas-l&D=0&H=0&O=T&T=1&P=21550.

Huang, Ya (2004). "Re: arrows." A SAS-L posting on Jun. 23, 2004. Archived at http://listserv.uga.edu/cgi-bin/wa?A2=ind0406D&L=sas-l&D=0&H=0&O=T&T=1&P=23531.

Rorie, Deena and Lynette Duncan (2003). "A Handy Use of the %LINE Annotate Macro." A Coders' Corner paper. Paper 83-28. *SUGI 28 Proceedings*. Archived at http://www2.sas.com/proceedings/sugi28/083-28.pdf.

Whitlock, Ian (2002). "Re: GOTO and LINK vs IF THEN DO." A SAS-L posting on Jul. 24, 2002. Archived at http://listserv.uga.edu/cgi-bin/wa?A2=ind0207D&L=sas-l&D=0&H=0&O=T&T=1&m=106219&P=28195.

NECESSARY REMARK

ACKNOWLEDGEMENTS

We would like to thank Carlos Tadeu Santos Dias for posting an interesting question on SAS-L (Dias(2004)). Ya(2004) replied with a quick solution, which is further developed into `%cyArrow` that we present here.

AUTHORS

Chang Y. Chung has friendly co-workers, who call him “Changrila Ding-Dong.” They are not SAS programmers, though. Chang hangs out at SAS-L, mostly causing trouble, but always having a great time. He can also be reached at chang_y_chung@hotmail.com.

Ya Huang works for Amylin Pharmaceuticals. He is a recipient of the “Most Valuable SAS-Ler” award.