SAS/GRAPH® SOFTWARE MEETS THE LOGO TURTLE
Michael Friendly, York University

Abstract
This talk describes a set of SAS macros which provide the ability to draw with the SAS/GRAPH® ANNOTATE facility using the “turtle-relative” drawing commands of the Logo language, rather than absolute X·Y coordinates. This makes drawing much easier for many types of figures.

Another key feature of Logo is recursion, which also simplifies many graphics procedures. Since the SAS data step has neither local variables, nor recursion, it is hard to some things that are very easy in Logo. However, SAS macro variables are local variables, and recursive macros which model Logo procedures can be defined. A variety of Logo-like designs carried out with SAS/G RAPH are illustrated.

Introduction
The annotate facility of SAS/G RAPH is a powerful tool for customizing graphs or for designing entirely new graphic displays. A good deal of this power stems from the various data reference systems, specified by the XSYS = and YSYS = annotate variables. The variables X and Y refer to coordinate locations in the display area, but the corresponding values of XSYS and YSYS determine whether the X, Y coordinates are interpreted as, for example, data values, screen cells, or percentage values.

On the other hand, some kinds of figures or shapes are particularly difficult to draw with the annotate facility. For example, to draw a pentagon or other polygon you must calculate the (x,y) coordinate of each point on the polygon using the SIN and COS functions, whose arguments must be specified in radians:

```
data pentagon;
xsys=-2; ysys=2;
do i = 1 to 5;
   angle = i/5 * (2*3.14159);
   x = 50 * cos (angle);
   y = 50 * sin (angle);
   if i=1 then function -HOVE-;
   else function = -DRAW-;
   output;
end;
proc ganno data=pentagon datasys;
```

This paper describes an approach to make it easier to draw these kinds of geometric figures in SAS/G RAPH by implementing the ideas of “turtle graphics” from the Logo language as SAS macros. From another perspective, this can also be viewed as an example of how extensions to SAS/G RAPH may be explored by incorporating ideas from other programming environments.

What is Logo?
Logo is a computer language designed to make computing easily accessible to young children (Papert, 1980) in an educational setting. Part of its appeal is an extremely simple yet flexible graphics component, called “turtle graphics”. However, Logo is actually a descendent of Lisp, and was designed to provide powerful programming features for experts as well as novices; see Friendly (1988).

Logo Turtle Graphics
Contrast the SAS.GRAPH annotate DATA step above with the following command in the Logo language:

```
REPEAT 5 [ FORWARD 50 RIGHT 72 ]
```

This command, entered interactively to the Logo system, causes a pentagon to be drawn on the screen. The Logo primitive FORWARD 50 (or BACK 50) moves a graphic cursor, called “the turtle” forward (or back) 50 “turtle steps” in whatever direction it is currently facing. (The Logo graphics area is approximately 280 by 240 turtle steps. The coordinates range from -120 to +120 in the horizontal direction and to +140 in the vertical direction.) The primitive command RIGHT 72 (or LEFT 72) rotates the turtle 72 degrees in a clockwise (or counter-clockwise) direction from its present orientation, but does not change the turtle’s position.

On the other hand, some kinds of figures or shapes are particularly difficult to draw with the annotate facility. For example, to draw a pentagon or other polygon you must calculate the (x,y) coordinate of each point on the polygon using the SIN and COS functions, whose arguments must be specified in radians:

```
data pentagon;
xsys='2'; ysys='2';
do i = 1 to 5;
   angle = i/5 * (2*3.14159);
   x = 50 * cos (angle);
   y = 50 * sin (angle);
   if i=1 then function = 'MOVE';
   else function = 'DRAW';
   output;
end;
proc ganno data=pentagon datasys;
```

The Logo turtle is considered to have a pen which can be lowered or raised. The primitive commands PENUP and PENDOWN control the turtle’s pen position; the command SETPC controls the color of ink in the pen. If the pen is down, the turtle draws in the current pen color when it moves; otherwise FORWARD and BACK cause the turtle to move without drawing a line. The REPEAT command is the Logo primitive for iteration. In the command

```
REPEAT 5 [ FORWARD 50 RIGHT 72 ]
```

the list of commands in square brackets is repeated 5 times.

Logo procedures. The commands to draw any figure can be defined as a new procedure with the TO command. For example, these statements define a procedure named PENTAGON.
Once a procedure has been defined, typing the name of the procedure causes the commands in the definition to be carried out.

Logo procedures can also be defined to take variable inputs, just like the Logo primitives `FORWARD` and `RIGHT`. This `PENTAGON` procedure draws a pentagon of any size:

```logo
TO PENTAGON :SIZE
  REPEAT 5 [ FORWARD :SIZE RIGHT 72 ]
END
```

In Logo, a variable argument to a procedure is represented by a variable name preceded by a colon, such as `:SIZE`, so if the procedure `PENTAGON 100` is invoked, the value 100 is substituted wherever `:SIZE` occurs.

Procedures can take any number of variable inputs. The procedure `POLY` draws a regular polygon of any `:SIZE` with any number of `:SIDES`:

```logo
TO POLY :SIDES :SIZE
  REPEAT :SIDES [ FORWARD :SIZE RIGHT 360/:SIDES ]
END
```

One of the powerful features of Logo, which makes it particularly easy for children to do interesting graphics, is this: Once a procedure has been defined, it can be used in other procedures exactly like the built-in primitive commands. For example, the procedure `SPIN.POLY` draws a series of `:NUMBER` polygons, rotating each one from the last:

```logo
TO SPIN.POLY :SIDES :SIZE :NUMBER
  REPEAT :NUMBER ( POLY :SIDES :SIZE
    FORWARD :SIZE/4 RIGHT 360/:NUMBER
  )
END
```

Recursion & local variables

Another powerful feature of Logo is recursion: procedures can do part of a task, then call themselves to complete the job. To support recursion, the variable inputs of a procedure are local variables: the values assigned to those variables in one invocation of a procedure have no effect on the values of those variables in any other procedure invocation.

The procedure `POLYSPI` draws a "polyspiral", a figure like a polygon, but with the length of each side decreasing until it becomes zero. The recursive way to do this in Logo is to draw one side, then call the same procedure to continue the same process:

```logo
TO POLYSPI :SIDE :ANGLE
  IF NOT :SIDE > 0 [STOP]
  FORWARD :SIDE RIGHT :ANGLE
  POLYSPI (:SIDE-1) :ANGLE
END
```

The operation of recursion in Logo can be shown by tracing the execution of the `POLYSPI` procedure, which causes the procedure to print the values of its input variables on each invocation. The "?" character is the Logo prompt.

```logo
?TRACE "POLYSPI
?POLYSPI 4 45
  --> Entering POLYSPI 4 45
  --> Entering POLYSPI 3 45
  --> Entering POLYSPI 2 45
  --> Entering POLYSPI 1 45
  --> Entering POLYSPI 0 45
  --> Leaving POLYSPI
  --> Leaving POLYSPI
  --> Leaving POLYSPI
  --> Leaving POLYSPI
  --> Leaving POLYSPI
-> Leaving POLYSPI
```

Each recursive call invokes a new copy of the `POLYSPI` procedure with its own, private values of the inputs `:SIZE` and `:ANGLE`. The previous copy is suspended, waiting for the new one to complete. When the value of `:SIZE` becomes zero, the `STOP` statement is executed, causing the inner-most version of `POLYSPI` to terminate. That allows the previous copy to continue, where the values of `:SIZE` and `:ANGLE` in that call are reinstated. But since `POLYSPI` is the last command in the procedure, that copy terminates too. The stacked recursive calls continue to unwind, until the initial call completes.

Implementing turtle graphics in SAS/GRAPH

In the Logo system, the turtle-relative drawing commands are implemented in such a way that the system maintains the turtle's current position and heading internally. When the command `FORWARD 50` is executed, the system simply calculates the new absolute `(x,y)` screen location using the standard trigonometric relations, and moves or draws to that position.

The Logo graphics primitives can be modelled in SAS/GRAPH with a set of SAS macros. The macros make use of two new data step variables, in addition to those used by the ANNOTATE facility:

- `PEN ('UP','DOWN')` determines whether the turtle moves or draws.
- `HEADING` turtle's current heading, in degrees, where 0 = NORTH and angles increase clockwise.

The basic Logo macros are:

- `xpenup` Lift PEN up, so turtle will move
- `xpendown` Put PEN down, so turtle will draw
- `xleft(degrees)` Turn left (counterclockwise)
- `xright(degrees)` Turn right (clockwise)
- `xforward(dist)` Move draw forward in current HEADING
- `xback(dist)` Move draw back in current HEADING
- `zinilogo` Initialize variables, move turtle home
- `xhome` Move draw to (0,0)

These macros are used in a DATA step to produce an ANNOTATE = data set.
The macros Zpenup, Zpendown, Zleft, and Zright simply set the values of the PEN and HEADING variables. Note that Zleft and Zright keep the HEADING variable in the range of 0 to 360 degrees.

Zmacro penup;
%--------------------------------------------;
% Put PEN UP, so turtle will move;
%-------------------------------------------~;
PEN="UP";
%mend penup;

Zmacro pendown;
%--------------------------------------------;
% Put PEN down, so turtle will draw;
%--------------------------------------------;
PEN="DOWN";
%mend pendown;

Zmacro left(degrees);
%--------------------------------------------;
% Turn LEFT by DEGREES (counterclockwise);
%--------------------------------------------;
HEADING=mod(HEADING-(degrees),360);
%mend left;

Zmacro right(degrees);
%--------------------------------------------;
% Turn RIGHT by DEGREES (clockwise);
%--------------------------------------------;
HEADING=mod(HEADING+(degrees),360);
%mend right;

All the work is done in the Zforward macro. For efficiency the macro recognizes the special cases of HEADING = 0, 90, 180, or 270 degrees. The Zback macro is implemented in terms of Zforward.

Zmacro forward(dist);
%--------------------------------------------;
% Move/Draw FORWARD in the current HEADING;
% X & Y are in whatever XSYS, YSYS system;
% is being used, but must be absolute;
% In turtle coordinates, HEADINGs go clockwise;
% go counterclockwise from East, so;
% HEADING = 90-angle;
% PEN determines whether to MOVE or DRAW;
%--------------------------------------------;
select (HEADING);
when ( 0 ) Y = Y + &dist;
when ( 90 ) X = X + &dist;
when ( 180 ) Y = Y - &dist;
when ( 270 ) X = X - &dist;
otherwise do; /* general case */
X=X+(&dist)*cos((90-HEADING)*atan(1)/45);
Y=Y+(&dist)*sin((90-HEADING)*atan(1)/45);
end;
#macro back(dist);
%--------------------------------------------;
% Move/Draw BACK in the current HEADING;
%-------------------------------------------~;
forward(-1*(&dist));
%mend back;

Two utility macros complete the basic set of Logo commands for SAS/GRAPH. Zhome moves the turtle to the "home" position, (0, 0). Zinilogo initializes the PEN and HEADING variables and starts the turtle at (0, 0).

Zmacro home;
%--------------------------------------------;
% Move/Draw to (0,0);
%--------------------------------------------;
X=0; Y=0;
if PEN = 'UP' then FUNCTION = 'MOVE';
else FUNCTION = 'DRAW';
output;
%mend home;

Zmacro inilogo;
%--------------------------------------------;
% Initialize logo variables & move turtle home;
% - use at start of data step, like dclanno;
%--------------------------------------------;
LENGTH PEN $; 4;
penup; Zhome; Zpendown;
HEADING=0;
%mend inilogo;

Using the Logo macros

The Logo macros are invoked in a DATA step to produce an ANNOTATE data set. The figure can then be drawn with PROC GANNO or PROC GSLIDE.

The example below draws a set of 12 stars equally spaced around a circle, giving the result shown in Figure 1. This example uses the data percentage coordinate system (XSYS='1'; YSYS='1';), so the plot is centered at (50, 50). The Logo macros can use any of the absolute coordinate systems.

%include LOGO ;
data example1;
xsys='1'; ysys='1';
inilogo;
x = 50; y=50;
function='MOVE'; output;
zpendown;
do i=1 to 12;
  Zforward(15);
do j=1 to 5;
  Zforward(20);
  Zright(144);
end;
Zback(15); Zright(36);
end;
proc ganno anno=example1;
Some sample SAS/GRAPH - Logo designs

As in Logo, the basic Logo macros can be used to define new procedures as macros. The possibilities are limited only by your imagination. For simplicity, I'll illustrate some variations and combinations of polygon designs. Here are SAS/GRAPH versions of the Logo POLY and POLYSPI procedures:

```
xmacro polys(sides,length);
  %* Move/Draw a polygon, the Logo way
  %*--------------------------------------------;
  do _i_ = 1 to &sides;
    %forward(&length);
    %right(360/(&sides));
  end;
  %end polys;

xmacro polyspi(length,angle,decrease);
  %*--------------------------------------------;
  %* Draw a polyspiral, decreasing side while &>0;
  %*--------------------------------------------;
  _len_ = &length;
  do while (_len_ > 0);
    %forward(_len_);
    %right(&angle);
    _len_ = _len_ - &decrease;
  end;
  %end polyspi;
```

Figure 2 shows a series of six polygons drawn with %polys. The DATA step which draws this figure is shown below: The program uses the SAS/GRAPH annotate macros, %system, %frame, %label, etc. in addition to the Logo macros.

```
data polyl;
  %dclanno;
  /* annotate macros */
  %system(2,2,4); /* data system */
  %frame(BLACK,1,1,EMPTY);
  %inilogo;
  Zmove(10,10); %pendown;
  do i=4 to 9;
    %polys(i, 15); %right(60);
  end;
  %label(8,30,'polys',RED,0,2.5,DUPLEX,5);
  proc ganno datasys anno=polyl;
```

The polygons drawn by %polys start and end at the turtle's current position. For use as a building block in larger designs, it is more useful to be able to draw a figure centered at the current position, and to be able to control the radius of a circumscribing circle, rather than the size of one side. The macro %cpoly provides these additional controls. The macro %spingon uses %cpoly to draw a series of rotated, centered polygons of decreasing radius.

```
xmacro cpoly(sides,radius);
  %*--------------------------------------------;
  %* Draw a centered polygon inscribed in a
  %* circle of given radius.
  %*--------------------------------------------;
  %move out to circumscribed circle;
  %penup; %home; %pendown;
  %forward(25);
  %polyspi(72,75,5);
  %right(72);
  end;
  %label(8,12,'POLYSPI',BLACK,0,0,1.5,DUPLEX,5);
  proc ganno datasys anno=poly2;
```

Figure 3 is drawn with %polyspi with this DATA step:

```
data poly2;
  %dclanno; %system(2,2,4);
  %frame(BLACK,1,1,EMPTY);
  %inilogo;
  do i=1 to 5;
    %penup; %home; %pendown;
    %forward(25);
    %polyspi(72,75,5);
    %right(72);
  end;
  %label(8,12,'POLYSPI',BLACK,0,0,1.5,DUPLEX,5);
  proc ganno datasys anno=poly2;
```

The polygons drawn by %polys start and end at the turtle's current position. For use as a building block in larger designs, it is more useful to be able to draw a figure centered at the current position, and to be able to control the radius of a circumscribing circle, rather than the size of one side. The macro %cpoly provides these additional controls. The macro %spingon uses %cpoly to draw a series of rotated, centered polygons of decreasing radius.

```
xmacro cpoly(sides,radius);
  %*--------------------------------------------;
  %* Draw a centered polygon inscribed in a
  %* circle of given radius.
  %*--------------------------------------------;
  %move out to circumscribed circle;
  %penup; %forward(90*(&sides-2)/&sides));
  ``
Xpolyspi design

Zpendown;
Z% polygon, side as function of radius;
Zpolys( &sides, (2* &radius)*sin( &pi/ &sides) ));
Z% turn & move back to starting point;
Zleft( 180 - (90* ( &sides - 2)/ &sides ));
Zpenup;
Zback(&radius);
Zpendown;
Zmend cpoly;

Zmacro spingon{ &sides, &rad, &decrease, &times, &turn);
Z%------------------------~-------------------;
Z% Centered polygons, of decreasing radius ;
Z%--------------------------------------------;
Zlen= &rad;
do _j_=l to &times;
Zcpoly( &sides, _len_) _j
Z%right(&turn) ;
Zlen=_len_ - &decrease;
end;
Zleft( &turn* &times );
Zmend spingon;

A design created with %spingon is shown in Figure 4.
This figure is drawn with the following statements:

data spin2;
Zdclanno; Zsystem(2,2,4);
Z%frame(BLACK,1,1,EMPTY);
Z%inilogo;
do H = 0, 100;
do V = 0, 100;
Zmove( H,V );
Zspingon( 4, 50,2,25,4 );
end; end;
Zmove( 50,50);
Zspingon( 4, 50,2,25,114); proc ganno datasys anno=spin2;

Turtle Text

The Logo macros can also be used to construct interesting graphic designs with text. The example below repeats a character string in a square spiral pattern, shown in Figure 5.

Zannocms;
data textl;
Zdclanno; Zsystem(2,2,4);
Z%inilogo;
Zstr="TURTLES DO IT RECURSIVELY.");
Zpenup; Zright(90);
Zcolor = 'BLACK';
do s=1 to 20;
Zdo i = 1 to &times;
Ztext = substr( Zstr, i, 1);
Zstr = substr( Zstr, 2) | substr( Zstr, 1, 1); angle= 90 - HEADING ; rot=angle ;
Zforward(11);
Zstyle = 'DUPLEX'; size=2.5;
Zfunction='LABEL'; output;
end;
Zright(90);
end;
proc ganno datasys anno=textl;

Recursive SAS/GRAPH macros

Recursion is possible in almost any programming language in which parameters to procedures or subroutines are local variables. In the SAS DATA step, all variables are global, and subroutines, called by the LINK statement do not allow parameters. However, in the SAS macro facility, macro parameters are local variables and other macro variables can be declared local to a macro with the ZLOCAL statement. This
TURTLES DO IT RECURSIVELY. TURTLES DO IT RECURSIVELY TURTLES DO IT RECURSIVELY TURTLES DO IT RECURSIVELY TURTLES DO IT RECURSIVELY TURTLES DO IT RECURSIVELY TURTLES DO IT RECURSIVELY TURTLES

Figure 5: Turtle Text design makes it possible to define recursive SAS macros which mirror Logo procedures.

The macro %repoly draws a set of recursive polygons. The "outer-level" of the procedure draws a centered polygon, like %zcpoly. However, at each vertex of this polygon, it calls itself to draw a new polygon centered at that vertex. The number of levels of recursion is specified by the depth parameter, and the macro keeps recursing until depth becomes zero. At each level, the size of the polygon is multiplied by the scale factor factor. Figure 6 shows a sample design drawn with %repoly.

Figure 6: %repoly( 6, 50, .5, 30, 3);

The state of each currently-running invocation and the values of local variables. SAS macros, on the other hand simply generate program text.

As a result, a macro like %repoly can easily cause the SAS supervisor to run out of memory or symbol table space. The macro call,

%repoly( 6, 50, .5, 30, 3);

for example, specifies a set of recursive hexagons to a depth of 3. At each of the six sides of the outer hexagon the macro calls itself with depth=2: %repoly( 6, 25, .5, 30, 2); and each of those calls results in a further recursive call with depth=1. That is about all the SAS supervisor can manage.

Limitations. While recursive macros can be written as illustrated, the SAS macro facility was never designed for this purpose. In languages that explicitly support recursion, the execution path of a recursive procedure is maintained in a "stack frame", an internal data structure that records the state of each currently-running invocation and the values of local variables. SAS macros, on the other hand simply generate program text.

As a result, a macro like %repoly can easily cause the SAS supervisor to run out of memory or symbol table space. The macro call,

%repoly( 6, 50, .5, 30, 3);

for example, specifies a set of recursive hexagons to a depth of 3. At each of the six sides of the outer hexagon the macro calls itself with depth=2: %repoly( 6, 25, .5, 30, 2); and each of those calls results in a further recursive call with depth=1. That is about all the SAS supervisor can manage.

Author's Address. For further information, contact:

Michael Friendly
Psychology Department, Rm 210 BSB
York University
Downsview, ONT, Canada M3J 1P3
BITNET: <FRIENDLY@YORKU.M1>