PROGRAM SUBMITTAL FORM

Complete this form using black ink or type (for duplicating purposes), and send with your program to the Computer Documents Library at your location or nearest you.

REQUIRED INFORMATION

Date 8/9/74

Program Name BE YAPP

Author(s) R.H. Bradley, B.N. Dickman

Dept. 9152 Ext. 7922 Loc. & Room RR 4c-120

Purpose of Program Plotting Package for use with the GSI-300 terminal under Unix.

Program Language C Machine UNIX

New Program _____ Modification of Existing Program _____

Documentation being Submitted Includes: MM for file

Binary Deck _____ Source Deck _____ Other _____

Compilation and/or Listing _____

Main Program _____ Subroutine _____ Function _____

COMPLETE IF APPLICABLE

Usage, Including Calling Sequence and Description of Variac

Bell Laboratories

subject: yapp (yet another plotting package)
case 39373-98

date: May 10, 1974
from: R. H. Bradley
B. N. Dickman

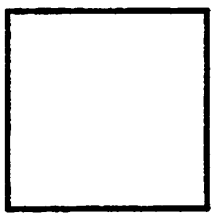
MEMORANDUM FOR FILE

1. Introduction

A plotting package is now available for use with the GSI-300 terminal under UNIX. As an example of its use, the following C program draws a square one inch on a side.

```
main()
    {
    openplot(); /* Default starting point is (0.0,0.0) in inches. */
    line(1.0,0.0);
    line(1.0,-1.0);
    line(0.0,-1.0);
    line(0.0,0.0);
    closeplot();
    }
```

The square that is drawn looks like the following:



2. The Plotting Functions

The plotting functions are listed below in the following sections in alphabetic order with brief descriptions of their use. Appendix I is a listing of the plotting functions.

```
cctrans(xfn,yfn)
    double(*xfn)(),(*yfn)();
```

may be used to compose coordinate transformation functions. Each of the coordinate transformation functions is passed two double precision arguments, x and y coordinates. Functions are composed in a "last specified first evaluated" order. A function may be deleted (last composed first deleted) by specifying

```
cctrans(DELETE);
```

The following examples illustrate the use of cctrans.

```
#include "common"
double xtr(), ytr();
double atr(), btr();
double sin(),cos();
/* a "generic" circle drawing program: */
circle()
    {
    cctrans(xtr,ytr);
    plotpen(UP);
    line(1.0,0.0);
    plotpen(DOWN);
    /* draw a line 2pi radians long. */
    line(1.0,6.283);
    }
double xtr(r,theta)
    double r,theta;
    {
    return(r*cos(theta));
    }
double ytr(r,theta)
    double r,theta;
    {
    return(r*sin(theta));
    }
/* using circle() to draw a circle of radius 2 about (3.,0.):
   (The origin is at the left margin, so that a circle cannot
   be drawn about it.) */
main()
    {
    openplot();
    precision(1);
    granularity(10);
    cctrans(atr,btr);
    circle();
    closeplot();
    }
double atr(x,y)
    double x,y;
    {return(2.*x+3.);}
double btr(x,y)
    double x,y;
    {return(2.*y);}
```

```
ctrans(xfn,yfn)
    double(*xfn)(),(*yfn)();
```

may be used to change the coordinate transformation function.

ctrans may also have SAVE or RESTORE as argument. (See section 3.) ctrans is the same as cctrans, except that ctrans is used to change a whole chain of compositions of coordinate transformations, while cctrans is used to add to or delete from the chain of coordinate transformations.

```
closeplot()
```

restores the terminal to its state before openplot().

```
dline(dx,dy)
    double dx,dy;
```

may be used to draw a line or move the plot pen. The parameters are in inches, and coordinate transformations which may be in effect are not applied. The pen stops dx inches to the right and dy inches above the current element position. The plotpen function determines whether the line is actually drawn. (If plot pen is UP, the primitive speeds up motion by using normal spacing instead of plot-mode spacing.)

```
granularity(action)
    int action;
```

where action is a positive integer, SAVE, or RESTORE, breaks a plot into isolated points such that no point is within action hundredths of an inch of the immediately preceding point. Default granularity is 0., although the GSI terminal will not produce points separated by less than 1/60 inch. SAVE and RESTORE are described in section 3.

```
line(x,y)
    double x,y;
```

is used to draw lines or move the plot pen. The pen stops at the int corresponding to the x-y coordinates specified in the parameters. Any coordinate transformations in effect are applied, and the result in inches is the actual end point. See also precision().

```
openplot()
```

puts the terminal into plot mode. It also initializes the state of the plot as follows:

```
granularity(0);
plotpen(DOWN);
plotchars(".");
pointsize(1);
precision(9999);
```

```
plotpen(action)
    int action;
```

where action is UP, DOWN, SAVE, or RESTORE, may be used to control whether the line specified in line() or dline() is actually drawn (plotpen(DOWN)), or the plot pen is just positioned at the end point of the line specified (plotpen(UP)). SAVE and RESTORE are described in section 3.

```
plotstate(action)
    int action;
```

where action is SAVE or RESTORE, may be used to save (action is SAVE) or restore (action is RESTORE) the state of the plot. (See also section 3.) The state of the plot is defined as the states produced by the plotpen, plotchars, pointsize, cctrans, granularity, precision, and ctrans functions.

```
pointsize(n)
    int n;
```

where n is a positive integer, SAVE, or RESTORE, may be used to set the size of the point used for plotting (and therefore the size of the line drawn). n is an integer from 1 to 1. 1 is the default. SAVE and RESTORE are described in section 3. (This function will be implemented for other values of n upon demand.)

```
plotchars(chars)
    char *chars;
```

where chars is a pointer to a character string, SAVE, or RESTORE, may be used to specify the character(s) to be used for each point. "." is the default. There is no limit on the number of characters that can be specified. The character string must, however, result in no net movement of the plot pen. SAVE and RESTORE are described in section 3.

```
precision(action)
    int action;
```

where action is a positive integer, SAVE, or RESTORE, generally must be used with non-linear coordinate transformations. It breaks each line into segments so that the interior of the line is also transformed. Therefore a single call to line() can result in a curve. The default is large segments, to avoid breaking up lines. The parameter is in hundredths of an inch and refers to the untransformed coordinates. For example, precision(50) breaks the line into half-inch segments. SAVE and RESTORE are described in section 3.

3. SAVE and RESTORE

granularity, plotpen, plotchars, pointsize, precision and ctrans may have as argument SAVE or RESTORE. SAVE pushes onto stacks the plot characters, the point size, the granularity, the precision, the coordinate transformation functions being used, or the state of the plot pen and sets the new state initially to be the same as the old. RESTORE pops the appropriate stack and restores the previous state of the plot for the appropriate function. The functions will return -1 on stack overflow or an attempt to restore past the end of the stack, zero otherwise.

4. Using the Plotting Functions

The plotting functions reside in library /usr/bnd/plot/p. The user must specify

```
#include "usr/bnd/plot/common"
```

in order to get certain "define" constants initialized correctly. To use the output of the plotting functions with nroff, see R. H. Bradley.

R. H. Bradley

B. N. Dickman

Distribution:
Messrs. J. F. Maranzano
9152 Members

APPENDIX I: LISTING OF THE PLOTTING FUNCTIONS

```
#define STACKSIZE 15
#define PTR -1
#define SAVE -2
#define RESTORE -3
#define UP -4
#define DOWN -5
#define ON -6
#define OFF -7
#define LEFT -8
#define RIGHT -9
#define DELETE -10

#define COMPOSEMAX 5 /* Maximum number of compositions allowed. */
struct xytrans  /*used by ctrans() and cctrans()*/
    {
    int composenum;
    double (*xtrans[COMPOSEMAX])();
    double (*ytrans[COMPOSEMAX])();
    };
int xplot,yplot;
double xcurrent,ycurrent;
int xlastg,ylastg;
double *ptrprecision,*ptrgranularity;
#include "common"
int otty[3], mtty[3];

closeplot()
    {
    /* Restore CR generated by LF. */
    stty(1,otty);
    plotmode(OFF);
    return(0);
    }

cctrans(xfn,yfn)
    double (*xfn)(),(*yfn)();
    {
    struct xytrans *ptr;
    int x;
    struct xytrans *ctrans();
    ptr = ctrans(PTR);
    x = xfn /*so that compares work correctly*/;
    if(x == DELETE)
        {
        if(ptr -> composenum == 0) return(-1) /* nothing to delete*/;
        (ptr -> composenum)-- ;
        return(0);
        }
    if(ptr -> composenum == COMPOSEMAX) return(-1) /*can't add any more*/;
    ptr -> xtrans[(ptr -> composenum)] = xfn;
    ptr -> ytrans[(ptr -> composenum)++] = yfn;
    return(0);
```

```
    }

cleanup()
    {
    closeplot();
    exit(2);
    }

struct xytrans *ctrans(xfn,yfn)
    double (*xfn)(),(*yfn)();
    {
    static struct xytrans transtack[STACKSIZE];
    static int stackindex;
    int x;
    int i;
    x = xfn /*so that comparisons work correctly*/;
    if(x == RESTORE)
        {
        /* Set number of compositions to zero. */
        if(stackindex == 0) return(-1) /*error: nothing to restore*/;
        stackindex--;
        return(0);
        }
    if(x == SAVE)
        {
        if(stackindex == STACKSIZE) return(-1) /* stack overflow*/;
        i = transtack[stackindex+1].composenum = transtack[stackindex].compos
        while(i--)
            {
            transtack[stackindex+1].xtrans[i]=transtack[stackindex].xtrans[i
            transtack[stackindex+1].ytrans[i]=transtack[stackindex].ytrans[i
            }
        stackindex++;
        return(0);
    }
    if(x == PTR)
        return(&(transtack[stackindex]));
    /* Set new coord. transformation. */
    transtack[stackindex].xtrans[0] = xfn;
    transtack[stackindex].ytrans[0] = yfn;
    transtack[stackindex].composenum = 1;
    return(0);
    }

genpt()
    {
    /* Worry about point size later. */
    printf("%s",plotchars(PTR));
    }

int *granularity(action)
    int action;
    {
    static int stackindex;
```

```
    static double agranularity[STACKSIZE];
    double deltaxy;
    if (action == SAVE)
        {
        if (stackindex == STACKSIZE)  return(-1);
        agranularity[stackindex + 1] = agranularity[stackindex++];
        ptrgranularity = &(agranularity[stackindex]);
        return (0);
        }
    if (action == RESTORE)
        {
        if (stackindex == 0)  return (-1);
        stackindex--;
        ptrgranularity = &(agranularity[stackindex]);
        return (0);
        }
    if (action == PTR) return (0);
    deltaxy = action;
    if (deltaxy <= 0.)  agranularity[stackindex] = 0.;
    else
        {
        deltaxy =/100.;
        agranularity[stackindex] = deltaxy * deltaxy;
        }
    ptrgranularity = &(agranularity[stackindex]);
    return (0);
    }

openplot()
    {
    /* Call cleanup in case of abort. */
    signal(2,cleanup);
    /* Turn off CR generated by LF. */
    gtty(1,otty);
    mtty[0] = otty[0];
    /* Also, turn off line feed delay. */
    mtty[2] = otty[2] & 077757 | 1 ;
    stty(1,mtty);
    plotmode(ON);
    precision(9999);
    granularity(0);
    xlastg = -500;
    ylastg = 49;
    /* Reset pen to origin. */
    plotpen(UP);
    line(0.0,0.0);
    plotchars(".");
    pointsize(1);
    plotpen(DOWN);
    return(0);
    }

char *plotchars(chars)
    char *chars;
```

```
    {
    static char *aplotchars[STACKSIZE];
    static int stackindex;
    if(chars == RESTORE)
        {
        if(stackindex == 0) return(-1);
        stackindex--;
        return(0);
        }
    if(chars == SAVE)
        {
        if(stackindex == STACKSIZE) return(-1);
        aplotchars[stackindex+1] = aplotchars[stackindex++];
        return(0);
        }
    if(chars == PTR)
        return(aplotchars[stackindex]);
    aplotchars[stackindex] = chars;
    return(0);
    }

int *plotmode(sw)
    {
    static int pmode;
    if (pmode == sw) return(-1);
    if(sw == PTR) return(&pmode);
    pmode = sw;
    /* Control F changes plot mode but does not print. */
    if(sw == ON) putchar(06);
    /* Control G exits plot mode but does not print. */
    if(sw == OFF) putchar('');
    return(0);
    }

plotpen(action)
    {
    static int aplotpen[STACKSIZE];
    static int stackindex;
    if(action == RESTORE)
        {
        if(stackindex == 0) return(-1);
        stackindex--;
        return(0);
        }
    if(action == SAVE)
        {
        if(stackindex == STACKSIZE) return(-1);
        aplotpen[stackindex+1] = aplotpen[stackindex++];
        return(0);
        }
    if(action == PTR) return(&(aplotpen[stackindex]));
    aplotpen[stackindex] = action;
    return(0);
    }
```

```
plotstate(action)
    int action;
    {
    return(plotpen(action) |
        plotchars(action) |
        pointsize(action) |
        precision(action) |
        granularity(action) |
        ctrans(action));
    }

pointsize(n)
    {
    static int apointsize[STACKSIZE],stackindex;
    if(n == RESTORE)
        {
        if(stackindex == 0) return(-1);
        stackindex--;
        return(0);
        }
    if(n == SAVE)
        {
        if(stackindex == STACKSIZE) return(-1);
        apointsize[stackindex+1] = apointsize[stackindex++];
        return(0);
        }
    if(n == PTR) return(&(apointsize[stackindex]));
    apointsize[stackindex] = n;
    return(0);
    }

int *precision(action)
    int action;
    {
    static int stackindex;
    static double aprecision[STACKSIZE];
    double deltaxy;
    if (action == SAVE)
        {
        if (stackindex == STACKSIZE)  return(-1);
        aprecision[stackindex + 1] = aprecision[stackindex++];
        ptrprecision = &(aprecision[stackindex]);
        return (0);
        }
    if (action == RESTORE)
        {
        if (stackindex == 0)  return (-1);
        stackindex--;
        ptrprecision = &(aprecision[stackindex]);
        return (0);
        }
    if (action == PTR) return (0);
    deltaxy = action;
    if (deltaxy <= 0.)  aprecision[stackindex] = 9999.;
```

```
    else    aprecision[stackindex] = deltaxy / 100.;
    ptrprecision = &(aprecision[stackindex]);
    return (0);
    }
#include "common"
double pldist();
line(x,y)
    double x,y;
    {
    double ax,ay,dx,dy,delx,dely;
    double deltaxy;
    int *plotpen();
    deltaxy = *ptrprecision;
    if (*(plotpen(PTR)) == DOWN)
        {
        ax = xcurrent;
        ay = ycurrent;
        dx = x - ax;
        dy = y - ay;
        if (dx < 0)
            {
            delx = -deltaxy;
            }
        else  delx = deltaxy;
        if (dy < 0)  dely = -deltaxy;
        else  dely = deltaxy;
        if (dy * dely <= dx * delx) /* then let x determine increment*/
            {
            if (dx != 0.)
                {
                dely = delx * dy / dx;
                if (dx >= 0)
                    {
                    while ((ax =+ delx) <= x)
                        {
                        ay =+ dely;
                        xline(ax,ay);
                        }
                    }
                else  /* negative motion */
                    {
                    while ((ax =+ delx) >= x)
                        {
                        ay =+ dely;
                        xline(ax,ay);
                        }
                    }
                }
            }
        else  /* let y determine the increment */
            {
            delx = dely * dx / dy;
            if (dy >= 0)
                {
```

```
                while ((ay =+ dely) <= y)
                    {
                    ax =+ delx;
                    xline(ax,ay);
                    }
                }
            else  /* negative y motion */
                {
                while ((ay =+ dely) >= y)
                    {
                    ax =+ delx;
                    xline(ax,ay);
                    }
                }
            }
        }
        xline(x,y);     /* finish the line */
    }
xline(x,y)
    double x,y;
    {
    struct xytrans *lptr;
    double dx,dy,currx,curry;
    double transx,transy,savetx;
    int tindex;
    struct xytrans *ctrans();
    lptr = ctrans(PTR);
    currx = (xplot) / 60.;
    curry = (yplot) / 48.;
    tindex = lptr -> composenum;
    transx = x;
    transy = y;
    while ( --tindex >= 0 )
        {
        savetx = transx;
        transx = (*(lptr -> xtrans[tindex])) (transx,transy);
        transy = (*(lptr -> ytrans[tindex])) (savetx,transy);
        }
    dline(transx - currx, transy - curry);
    xcurrent = x;
    ycurrent = y;
    }

dline(dx,dy)
    double dx,dy;
    {
    int px,py;
    if (dx > 0.)  px = 60. * dx + .5;
    else          px = 60. * dx - .5;
    if (dy > 0.) py = 48. * dy + .5;
    else          py = 48. * dy - .5;
    movepen(px,py);
    }
```

```
char plotmove[4] {10,11,8,' '};  /*down,up,left,right*/

#define YES 1
#define NO 0
int pointmoved YES;
 movepen(pdx,pdy)
    int pdx,pdy;
    {
    int state;
    double gran;
    int hor;
    int vert;
    double exact,tan,more,less;
    int dx,dy,md,ld;
    int moreinc;
    int lessinc;
    int mbiginc;
    int lbiginc;
    int *plotpen();
    gran = *ptrgranularity;
    state = *(plotpen(PTR));
    if ((state) == DOWN && pointmoved == YES)
        {
        if (gran > 0)
            {
            if (gran <= pldist())
                {
                genpt();
                xlastg = xplot;
                ylastg = yplot;
                }
            }
        else genpt();
        }
    dx = pdx;
    dy = pdy;
    moreinc = 0;
    lessinc = 0;
    if (dx > 0)  hor = RIGHT;
     else {
            hor = LEFT;
            dx = -dx;
        }
    if (dy > 0)  vert = UP;
     else {
            vert = DOWN;
            dy = -dy;
         }
    if (dx < dy) {
            more = dy;
            less = dx;
            md = vert;
            ld = hor;
            mbiginc = 8;
```

```
            lbiginc = 6;
        }
      else {
            more = dx;
            less = dy;
            md = hor;
            ld = vert;
            mbiginc = 6;
            lbiginc = 8;
        }
    if (more > 0)  {
            pointmoved = YES;
            tan = (10 * less) / more;
            if ((state) == UP) {
                    plotmode(OFF);
                    while (moreinc <= (more - mbiginc)) {
                        genmove(md);
                        moreinc =+ mbiginc;
                        }
                    while (lessinc <= (less - lbiginc)) {
                        genmove(ld);
                        lessinc =+ lbiginc;
                        }
                    plotmode(ON);
                    while (moreinc++ < more) genmove(md);
                    while (lessinc++ < less) genmove(ld);
                }
            plotmode(ON);
            while (moreinc++ < more) {
                genmove(md);
                exact = moreinc * tan;
                if (exact >= (lessinc + 5)) {
                    genmove(ld);
                    lessinc =+ 10;
                    }
                if ((state) == DOWN)
                    {
                    if (gran > 0)
                        {
                        if (gran <= pldist())
                            {
                            genpt();
                            xlastg = xplot;
                            ylastg = yplot;
                            }
                        }
                    else  genpt();
                    }
                }
            }
    else pointmoved = NO;
    }

genmove(i)
```

```
{
    int *plotmode();
    int pmstate;
    pmstate = *(plotmode(PTR));
    switch(i)
        {
        case UP: i=1;
            if (pmstate == ON) yplot =+ 1;
            else yplot =+ 8;
            break;
        case DOWN: i=0;
            if (pmstate == ON) yplot =- 1;
            else  yplot =- 8;
            break;
        case LEFT: i=2;
            if (pmstate == ON) xplot =- 1;
            else  xplot =- 6;
            break;
        case RIGHT: i=3;
            if (pmstate == ON) xplot =+ 1;
            else  xplot =+ 6;
            break;
        /*error default to be supplied*/
    default:
    {
    plotmode(OFF);
    printf("GENMOVE system error");
    }
        }
        putchar(plotmove[i]);
 }
plprint(chars)
    char *chars;
    {
    char *px;
    px = chars;
    plotmode(OFF);
    putchar(' '); /* spa
    while(*chars != '0')
        {
        putchar(*(chars++));
        }
    while(chars-- > px)
        putchar('');  /* backspace */
    putchar('');/*two backspaces*/
    }

double pldist()
    {
    double xdistp,ydistp;
    xdistp = xlastg - xplot;
    xdistp =/ 60.;
    ydistp = ylastg - yplot;
    ydistp =/ 48.;
```

```
return(xdistp*xdistp + ydistp*ydistp);
}
```