

**Graphics on the HP LaserJet  
Via C Programs on a Linux Machine**

Michael E. Taylor

## Contents

1. Printing ASCII characters via C
2. Sample graphics program using print.c
3. The contents of print.c
4. Mandelbrot set program
5. Lines and curves
6. More on the PCL5 programming language
7. Color graphics

## Introduction

These notes describe how to write C programs that produce graphical output on a Hewlett-Packard LaserJet printer. In particular we discuss how one can include the file "print.c", described in detail in §3, and then write C programs that use the following commands to produce graphical output on a LaserJet printer for which the printer command language PCL3 or PCL5 is available. Here are the main commands:

```
GrInit
SetCursor
PrintPic
GrEnds
FormFeed
Finish
```

And there are some additional commands:

```
PutRow
PrintRow
BWRow
```

In the sections that follow we shall specify the effects of these commands and illustrate them with several complete programs, accompanied by the graphical output.

We briefly outline the effect of these commands here. The command 'GrInit' resets the printer and prepares it to accept graphical input. The 'SetCursor' command specifies a point on the paper where printing will start. More precisely, `SetCursor(a,b)` has the printing start  $a$  inches from the left edge of the paper and  $b$  inches below the top edge of the paper.

The 'PutRow' command prepares the printer to accept an array of graphics data. It occurs as an ingredient in both the 'PrintRow' command and the 'BWRow' command. Both these latter commands print a 6.4 inch horizontal line of dots, arising either from a 240 element array or a 1920 element array of integers. The command 'BWRow' is really the workhorse for most of the graphics we produce on the printer with our C programs. But the command most commonly used in a program discussed in these notes is 'PrintPic,' which executes the BWRow command repeatedly.

The idea is that we set up a 1920 by  $N$  array of integers in RAM. Typical values of  $N$  are 1440 and 2400. We define a correspondence between a rectangular region in the plane and this rectangular array. Then we print out  $N$  adjacent horizontal lines, each 6.4 inches long. Each nonzero element of the array yields a black dot. We skip down  $1/300$  inch from line to line.

The ‘GrEnd’ command ends graphics; you usually don’t need it. Finally, the ‘FormFeed’ command causes the page produced to be ejected from the printer, and the ‘Finish’ command resets the printer.

The commands above work with the printer once we have out  $1920 \times N$  array set up in RAM. If the array is called `pix`, the nonzero elements of `pix[j][k]` define where black dots are printed, as mentioned above. The following commands are helpful for drawing lines and curves without having to laboriously set various elements `pix[j][k]=1` in a program:

```
lline
mline
sline
mcurve
scurve
Box
MBox
```

In §1 we discuss a program, `sieve.c`, which simply writes ASCII characters to the printer. This program does not use any of the graphics commands mentioned above, but it does give one a first brush with using the printer.

In §2 we present a program, `sine.c`, which uses the graphics commands in the first list to produce a graph of  $y = \sin x$ , over the range  $-2\pi \leq x \leq 2\pi$ . In §3 we list the file "`print.c`", used in the program `sine.c`, which contains the commands in the first two lists above, and briefly discuss how they work. In §4 we set down a program that draws a part of the Mandelbrot set, and also uses the graphics commands in the first list.

In §5, we discuss the line, curve, and box commands in the third list above, present the file "`draw.c`", which contains them, and also present several programs, `sine2.c`, `sine3.c`, `boxes.c`, which make use of these commands.

Section 6 has some further general information on the PCL5 programming language, which one can use to write further C functions to use for graphics on HP LaserJet printers. Still further information can be found in books listed in the references.

Section 7 deals with color graphics, discussing programs that include "`color-print.c`" and "`colordraw.c`", for use on a color HP InkJet printer, accepting PCL3 commands. There are versions of the commands in the first and third lists above, described in this section.

## 1. Printing ASCII characters via C

In this first section we discuss how to print out text in the form of ASCII characters. A program to do this is identical to a program that prints such characters on the computer screen; the difference arises in how it is run.

Consider the following program, `sieve.c`, which prints out the prime numbers less than 1000. On a Linux machine, you compile this by typing

```
gcc sieve.c -o sieve
```

Once it is compiled, you can run it by typing

```
./sieve
```

and then the prime numbers less than 1000 are displayed on the computer screen.

If instead you type

```
./sieve | lpr
```

then the output is redirected to the printer. Here is the program:

```
/* sieve.c */

#include <stdio.h>

main()
{
    int r[1001];
    int j,k,m,n;

    for (j=0;j<=1000;j++) r[j]=j;

    for (m=2;m<=33;m++) {
        if (r[m] !=0) {
            for (k=m;k<=1000/m;k++) r[k*m]=0;}
    }

    n=0;
    printf("\n");
    printf("\nPrimes via the Sieve");
    printf("\n");
    printf("\n");
    for (j=2;j<=1000;j++) {
        if (r[j] !=0) {
            n=n+1;
            printf("%5d",r[j]);
```

```

        if (n % 15 == 0) printf("\n");
    }
}

```

We mention that a new line is formed after every 15 primes, using `printf("\n")`. When this program is run, the printed output looks like this:

#### Primes via the Sieve

```

  2   3   5   7  11  13  17  19  23  29  31  37  41  43  47
 53  59  61  67  71  73  79  83  89  97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
199 211 223 227 229 233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349 353 359 367 373 379
383 389 397 401 409 419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541 547 557 563 569 571
577 587 593 599 601 607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727 733 739 743 751 757 761
769 773 787 797 809 811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919 929 937 941 947 953 967 971 977
983 991 997

```

## 2. Sample graphics program using print.c

The program `sine.c` listed below draws a graph of  $y = \sin x$ , for  $-2\pi \leq x \leq 2\pi$ . The program uses a number of ingredients that are contained in the file `print.c`. After listing the program, we will discuss various commands in it, and present a copy of the output. We remark that the program is compiled on a Linux machine using

```
gcc sine.c -lm -o sine
```

The `-lm` calls up a math library to evaluate  $\sin x$ . As in §1, to produce output on a LaserJet printer, one runs this program by typing

```
./sine | lpr
```

Here is the program:

```
/* sine.c */

#include <stdio.h>
#include <math.h>

#define MAXK 1440

#include "print.c"

main()
{
    int i,j,k;
    char pix[1920][1440];
    double pi,y,fx,fy;

    pi=3.14159265358979;
    fx=2*pi/900;
    fy=1.0/680;

    for (k=0;k<=1439;k++) {
        for (j=0;j<=1919;j++) pix[j][k]=0;
    }

    for (j=0;j<=1919;j++) pix[j][0]=1;
    for (j=0;j<=1919;j++) pix[j][1439]=1;
    for (k=0;k<=1439;k++) pix[0][k]=1;
    for k=0;k<=1439;k++) pix[1919][k]=1;
```

```

for (j=50;j<=1870;j++) pix[j][720]=1;
for (k=40;k<=1400;k++) pix[960][k]=1;

for (j=120;j<=3720;j++) {
    y=sin((0.5*j-960.0)*fx);
    k=720-(y/fy);
    i=j/2;
    pix[i][k]=1;
}

GrInit;
PrintPic(pix);

SetCursor(3.8,1.7);
printf("y");
SetCursor(7.2,3.7);
printf("x");
SetCursor(3.7,6.7);
printf("y = sin x");

FormFeed;
Finish;
}

```

In addition to including `<stdio.h>` and `<math.h>` at the beginning of this program, we also include the file `"print.c"`, which will be discussed in detail in §3.

We discuss the workings of this program. The array `pix[1920][1440]` is set up to correspond to the rectangle

$$\mathcal{R} = \left\{ (x, y) : -2\pi \frac{960}{900} < x < 2\pi \frac{960}{900}, -\frac{720}{680} < y < \frac{720}{680} \right\},$$

via an affine transformation:

$$x_j = (j - 960)f_x, \quad y_k = (720 - k)f_y,$$

with  $f_x = 2\pi/900$ ,  $f_y = 1/680$ , so  $(x_j, y_k)$  ranges over the rectangle  $\mathcal{R}$  as  $(j, k)$  ranges over  $[0, \dots, 1919] \times [0, \dots, 1439]$ .

Commands involving `pix[j][0]=1 ... pix[1919][k]=1` serve to put a box about this region. The x-axis and y-axis are then set up in `pix[j][720]=1` and `pix[960][k]=1`.

The next section of code sets `pix[i][k]=1` when  $(x_i, y_k)$  lies on (or at least very near) the graph of  $y = \sin x$ , under the correspondence of  $i$  with  $x_i$  and  $k$  with  $y_k$  indicated above.

The effect of the code described so far in this program is to set up the array `pix`. This array is to yield a graph on a sheet of paper, defined by commands starting with `GrInit`, which prepares the printer to accept graphical input. The work of sending this data to the printer is done by

```
PrintPic(pix);
```

The workings of this command will be discussed in §3. We mention that all graphical arrays such as `pix` are  $1920 \times N$  arrays of characters. In this case,  $N = 1440$ . The quantity  $N$ , denoted `MAXK` here, must be available to the function `Printpic`. Hence we needed to put the line

```
#define MAXK 1440
```

above the line `#include "print.c"`.

Once this graphics data is communicated, a few characters are printed, such as the location of the x-axis, the y-axis, and a caption: “`y = sin x`”. The command `SetCursor(3.8,1.7)`, which precedes `printf("y")`, causes the character `y` to be printed in a field whose lower left corner is 3.8 inches from the left edge of the paper and 1.7 inches below the top edge of the paper.

The last commands of the program are ‘`FormFeed`’, which ejects the page, and ‘`Finish`’, which resets the printer.

### 3. The contents of print.c

As we have stated, the file `print.c` contains the definitions and functions that make the graphics commands given in the first two lists in the introduction work. Here is this file:

```

/* print.c */

#define FormFeed printf("%c",12)

#define GrEnds printf("%c*rB",27)

#define GrInit printf("%cE%c&l1H%c&l0%c*p0X%c*p0Y \
    %c*t300R",27,27,27,27,27,27)

#define Finish printf("%c&l0%c(8U%c(sp10h12vsb3T%c&l1H", \
    27,27,27,27)
/* End printing with: GrEnds, FormFeed, Finish */

#define PutRow printf("%c*r1A%c*b%dW",27,27,240)
/* enter raster graphics mode and prepare to transfer */
/* 240 bytes of data */

#define SetCursor(a,b) printf("%c&a%6.1fh %6.1fV", \
    27,720*(a)-180,720*(b)-360)
/* Cursor position set at a inches from left edge, */
/* b inches below top edge */

void PrintRow(double a, double b, int d[240])
{
    int j;
    SetCursor(a,b);
    PutRow;
    for (j=0;j<=239;j++) putchar(d[j]);
}
/* draws a line using 240 bytes of data */
/* starting a inches from left edge, */
/* b inches below top edge */

```

```

void BWRow(double a, double b, int p[1920])
{
    int j;
    int q[1920];
    int d[240];
    for (j=0;j<=1919;j++) q[j]=0;
    for (j=0;j<=1919;j++) {
        if (p[j] != 0) q[j]=1;
    }
    for (j=0;j<=239;j++) {
        d[j]=128*q[8*j]+64*q[8*j+1]+32*q[8*j+2]+16*q[8*j+3]
            +8*q[8*j+4]+4*q[8*j+5]+2*q[8*j+6]+q[8*j+7];
    }

    SetCursor(a,b);
    PutRow;
    for (j=0;j<=239;j++) putchar(d[j]);
}
/* draws a line using 1920 data points */
/* line starts a inches from left edge, */
/* b inches below top edge */

void PrintPic(char g[1920][MAXK])
{
    int j,k;
    double ep;
    int dat[1920];

    ep=1.0/300;
    for (k=0;k<=MAXK-1;k++) {
        for (j=0;j<=1919;j++) dat[j]=g[j][k];
        BWRow(1.0,1.5+k*ep,dat);
    }
}

void RmFont(int k)
{
    printf("%c(8U",27);
}

```

```
    printf("%c(s1p%dv0s0b4101T",27,k);
}
```

We discuss the contents of this file. The guts in all cases are ‘printf’ statements, addressing the Hewlett-Packard Laserjet printer in the PCL5 printer language.

We give a brief description of each of the PCL5 printer language commands used above, listing it first as written in the program, then as written in LaserJet manuals, and then stating its effect. We will give a further discussion of the PCL5 printer language in §6.

The ‘escape’ command, sent to the printer as character number 27 (1B in hex code), tells the printer that a PCL5 command is to follow.

These six commands make up GrInit:

"%cE",27	$E_cE$	reset printer
"%c&l1H",27	$E_c&l1H$	select paper feed from tray
"%c&l0",27	$E_c&lO$	reset printer cursor position (O=‘oh’)
"%c*p0X",27	$E_c*p0X$	cursor position:0 dots horiz. (0=‘zero’)
"%c*p0Y",27	$E_c*p0Y$	cursor position:0 dots vert.
"%c*t300R",27	$E_c*t300R$	300 dpi resolution

The SetCursor command is used to set the cursor position. Its structure is the following:

```
"%c&a%6.1fh %6.1fV",27,X,Y     $E_c&a\#h\#V$ 
```

Here, the symbols # stand for real numbers, formatted like 1234.5, which specify, in ‘decipoints,’ the horizontal and vertical components of the cursor position. If X is  $x$ , the horizontal position of the cursor is set at

$$\frac{1}{4} \text{ in} + x \text{ decipt} = \frac{1}{4} + \frac{x}{720} \text{ inches}$$

to the right of the left edge of the paper. If Y is  $y$ , the vertical position of the cursor is set at

$$\frac{1}{2} \text{ in} + y \text{ decipt} = \frac{1}{2} + \frac{y}{720} \text{ inches}$$

below the top edge of the paper. In SetCursor(a,b), we have  $a$  and  $b$  measured in inches, and  $X = 720a - 180$ ,  $Y = 720b - 360$ .

A certain border of the page is unprintable; the border consists of all points within 1/4 inch of the left or right edge of the paper, and all points within 1/2 inch of the top or bottom edge.

The command ‘PutRow’ consists of the following two parts:

"%c*r1A",27	$E_{cr1A}$	enter raster graphics mode; start graphics at current cursor
-------------	------------	---

`"%c*b%dW",27,240`     $E_c*b240W$     transfer 240 bytes of graphic data

The data transfer is done in ‘PrintRow.’ Each byte of data causes a pattern of dots to be printed. What is printed is the following. Say the byte has value  $m \in [0, 255]$ . In a sequence of positions of 8 dots starting at the position the cursor is set at, there are printed dots precisely in the positions where the binary expansion of  $m$  contains a 1.

The function ‘BWRow’ takes an array of 1920 integers and produces an array of 240 bytes, which are then sent to the printer as in ‘PrintRow.’ More precisely, the effect of `BWRow(a,b,dat)` is to print a 6.4 inch horizontal row of dots, starting at a position  $a$  inches from the left of the edge of the paper and  $b$  inches below the top edge, with data given by the 1920-element array  $dat$ .

While ‘BWRow’ can be used directly in programs, it is typically convenient to use ‘PrintPic’, which converts a  $1920 \times N$  array (say `pix`) to print by executing  $N$  `BWRow` commands. More precisely, for each  $k \in [0, \dots, N - 1]$ , a 1-dimensional array `dat[j]=pix[j][k]` of data is fed into a line of output. The command

`BWRow(1.0,1.5+k*ep,dat)`

has this data printed on a horizontal line, 6.4 inches long, positioned at one inch from the left edge of the paper and  $1.5 + \varepsilon k$  inches below the upper edge of the paper. Here  $\varepsilon = 1/300$ , since we are using a 300 dots per inch printer.

`GrEnds` is the command to end graphics:

`"%c*rB",27`     $E_c*rB$

When the printing of graphics is complete, there is a form feed command:

`"%c",12`

the character number 12 (i.e., 0C in hex code). ‘Finish’ contains four commands to reset the printer:

<code>"%c&amp;l0",27</code>	$E_c&l0$	reset printer cursor position (O=‘oh’)
<code>"%c(8U",27</code>	$E_c(8U$	symbol set: Roman-8
<code>"%c(sp10h12vsb3T",27</code>	$E_c(sp10h12vsb3T$	reset primary font values
<code>"%c&amp;l1H",27</code>	$E_c&l1H$	select paper feed from tray

#### 4. Mandelbrot set program

Here we discuss a program that produces a picture of a small portion of the Mandelbrot set.

The Mandelbrot set  $\mathcal{M}$  is an immensely complicated set in the complex plane, but it has the following simple definition. Given  $c \in \mathbb{C}$ , we define

$$(4.1) \quad \Phi_c : \mathbb{C} \longrightarrow \mathbb{C}, \quad \Phi_c(z) = z^2 + c.$$

We consider the sequence of points in  $\mathbb{C}$ :

$$(4.2) \quad \xi_1(c) = c, \quad \xi_2(c) = \Phi_c(\xi_1(c)), \dots, \xi_{j+1}(c) = \Phi_c(\xi_j(c)), \dots$$

One says  $c \in \mathcal{M}$  if and only if  $\{\xi_j(c) : j \in \mathbb{Z}^+\}$  is *bounded* in  $\mathbb{C}$ . It is not hard to show that

$$(4.3) \quad c \notin \mathcal{M} \iff \text{some } |\xi_j(c)| > 2.$$

In a program to draw  $\mathcal{M}$ , we pick a “fairly large” number  $N$ . In `mndbt.c`, which will be listed below, we set  $N = 850$ . We consider a rectangular array of complex numbers, such as

$$(4.4) \quad c_{jk} = z_0 + \frac{1200 - k}{960}\ell + \frac{960 - j}{960}\ell i, \quad 0 \leq j \leq 1919, \quad 0 \leq k \leq 2399,$$

to produce a picture on a sheet of paper of a rectangular region

$$(4.5) \quad \mathcal{R} = \left\{ z = x + iy \in \mathbb{C} : x_0 - \frac{5}{4}\ell \leq x \leq x_0 + \frac{5}{4}\ell, \quad y_0 - \ell \leq y \leq y_0 + \ell \right\},$$

with  $z_0 = x_0 + iy_0$ . We declare that a complex number  $c = c_{jk}$  seems to be in  $\mathcal{M}$  provided  $|\xi_j(c)| \leq 2$  for all  $j \leq N$ . Then we blacken the dot associated to  $c_{jk}$  black, i.e., we set `pix[j][k]=1`.

Here is the program `mndbt.c`.

```
/* mndbt.c */

#include <stdio.h>

#define MAXK 2400
```

```

#include "print.c"

main()
{
    int j,k,m;
    char pix[1920][2400];
    double a,b,l,x,y,x1,u,v;

    a=-0.7454285;
    b=0.1130087;
    l=0.0000125;

    for (k=0;k<=2399;k++) {
        for (j=0;j<=1919;j++) pix[j][k]=0;
    }

    for (j=0;j<=1919;j++) pix[j][0]=1;
    for (j=0;j<=1919;j++) pix[j][2399]=1;
    for (k=0;k<=2399;k++) pix[0][k]=1;
    for (k=0;k<=2399;k++) pix[1919][k]=1;

    for (k=0;k<=2399;k++) {
        for (j=0;j<=1919;j++) {
            x=a+l*(1200-k)/960;
            y=b+l*(960-j)/960;
            u=x;
            v=y;
            m=1;
            while ((x*x+y*y<=4) & (m<850)) {
                x1=x;
                x=x*x-y*y+u;
                y=2*x1*y+v;
                m=m+1;
            }
            if (m>800) pix[j][k]=1;
            if ((m<400) & (m>250)) pix[j][k]=1;
        }
    }

    GrInit;
    PrintPic(pix);

    FormFeed;
    Finish;
}

```

}

Here, the center of the displayed rectangular area is

$$(4.6) \quad z_0 = -0.7454285 + 0.113087 i,$$

and the quantity  $\ell$  in (4.4)–(4.5) is

$$(4.7) \quad \ell = 0.0000125 = \frac{1}{8} \cdot 10^{-4}.$$

## 5. Lines and curves

As indicated in the introduction, here we describe the function of the following commands, and illustrate their use in some programs:

```
lline(p1,q1,p2,q2)
mline(p1,q1,p2,q2)
sline(p1,q1,p2,q2)
mcurve(a1,b1,p1,q1,p2,q2,a2,b2)
scurve(a1,b1,p1,q1,p2,q2,a2,b2)
Box(p1,q1,p2,q2)
MBox(p1,q1,p2,q2)
```

Briefly, `lline(p1,q1,p2,q2)` draws a line from the point  $(p_1, q_1)$  to  $(p_2, q_2)$ . This is done in `pix`-space, so  $p_j$  are integers in  $[0, \dots, 1919]$  and  $q_j$  are integers in  $[0, \dots, N - 1]$ , given that `pix` is a  $1920 \times N$  array. The commands involving `mline` and `sline` have the same effect, but they typically are applied to shorter lines.

The command `mcurve(a1,b1,p1,q1,p2,q2,a2,b2)` draws a curve in `pix`-space from  $(p_1, q_1)$  to  $(p_2, q_2)$ , interpolating in a smooth way via a cubic polynomial the piecewise linear connection from  $(a_1, b_1)$  to  $(p_1, q_1)$  to  $(p_2, q_2)$  to  $(a_2, b_2)$ . The command `scurve` has a similar effect, for shorter curves.

The command `Box(p1,q1,p2,q2)` draws a rectangle with sides that are horizontal and vertical, whose upper left corner is  $(p_1, q_1)$  and whose lower right corner is  $(p_2, q_2)$ , assuming  $p_1 < p_2$  and  $q_1 < q_2$ .

The first illustration of the use of the ‘line’ commands is `sine2.c`. We present this program, and then show the file `"draw.c"`, containing the various commands listed above.

```
/* sine2.c */

#include <stdio.h>
#include <math.h>

#define MAXK 1440

#include "print.c"
#include "draw.c"

char pix[1920][1440];
```

```

main()
{
    int i,j,k,jj,kk;
    double pi,y,fx,fy;

    extern char pix[1920][1440];

    pi=3.14159265358979;
    fx=2*pi/900;
    fy=1.0/680;

    for (k=0;k<=1439;k++) {
        for (j=0;j<=1919;j++) pix[j][k]=0;
    }

    lline(0,0,1919,0);
    lline(0,0,0,1439);
    lline(1919,0,1919,1439);
    lline(0,1439,1919,1439);

    lline(50,720,1870,720);
    lline(960,40,960,1400);

    for (j=60;j<=1860;j=j+60) {
        y=sin((j-960)*fx);
        k=720-(y/fy);
        if (j > 60) mline(jj,kk,j,k);
        jj=j;
        kk=k;
    }

    GrInit;
    PrintPic(pix);

    SetCursor(3.8,1.7);
    printf("y");
    SetCursor(7.2,3.7);
    printf("x");
    SetCursor(3.7,6.7);
    printf("y = sin x");

    FormFeed;
    Finish;
}

```

Here is the file "draw.c".

```

/* draw.c */

/* #define MAXK in your program */

#define lline(p1,q1,p2,q2) line(3500,p1,q1,p2,q2)
#define mline(p1,q1,p2,q2) line(500,p1,q1,p2,q2)
#define sline(p1,q1,p2,q2) line(50,p1,q1,p2,q2)

void line(int length, int p1, int q1, int p2, int q2)
{
    int i,j,k;
    double lng;
    extern char pix[1920][MAXK];
    lng=1.0*length;
    for (i=0;i<=length;i++) {
        j=(1.0*i*p1+(lng-1.0*i)*p2)/lng;
        k=(1.0*i*q1+(lng-1.0*i)*q2)/lng;
        pix[j][k]=1;
    }
}

#define mcurve(a1,b1,p1,q1,p2,q2,a2,b2) \
    curve(500,a1,b1,p1,q1,p2,q2,a2,b2)

#define scurve(a1,b1,p1,q1,p2,q2,a2,b2) \
    curve(50,a1,b1,p1,q1,p2,q2,a2,b2)

void curve(int length, int a1, int b1, int p1, int q1,
           int p2, int q2, int a2, int b2)
{
    int i,j,k;
    double lng,s1,s2,c0,c1,c2,c3,d0,d1,d2,d3,t;
    extern char pix[1920][MAXK];

    lng=1.0*length;

```

```

s1=0.5*(p2-a1);
s2=0.5*(a2-p1);
c0=1.0*p1;
c1=s1;
c2=3.0*(p2-p1)-2*s1-s2;
c3=2.0*(p1-p2)+s1+s2;

```

```

s1=0.5*(q2-b1);
s2=0.5*(b2-q1);
d0=1.0*q1;
d1=s1;
d2=3.0*(q2-q1)-2*s1-s2;
d3=2.0*(q1-q2)+s1+s2;

```

```

for (i=0;i<=length;i++) {
    t=i/lng;
    j=c0+t*(c1+t*(c2+t*c3));
    k=d0+t*(d1+t*(d2+t*d3));
    pix[j][k]=1;
}

```

```

}

```

```

void Box(int p1, int q1, int p2, int q2)
{
    lline(p1,q1,,p2,q1);
    lline(p1,q1,p1,q2);
    lline(p2,q1,p2,q2);
    lline(p1,q2,p2,q2);
}

```

```

void MBox(int p1, int q1, int p2, int q2)
{
    mline(p1,q1,p2,q1);
    mline(p1,q1,p1,q2);
    mline(p2,q1,p2,q2);
    mline(p1,q2,p2,q2);
}

```

As a graph of  $y = \sin x$ , the last graph is rather pathetic, with its craggy angles. The following produces a much better graph of this function, evaluating the sine function at the same 31 points. We use the 'mcurve' command.

```

/* sine3.c */

#include <stdio.h>
#include <math.h>

#define MAXK 1440

#include "print.c"
#include "draw.c"

char pix[1920][1440];

main()
{
    int i,j,k,jj,kk;
    int xa[31],ya[31];
    double pi,y,fx,fy;

    extern char pix[1920][1440];

    pi=3.14159265358979;
    fx=2*pi/900;
    fy=1.0/680;

    for (k=0;k<=1439;k++) {
        for (j=0;j<=1919;j++) pix[j][k]=0;
    }

    lline(0,0,1919,0);
    lline(0,0,0,1439);
    lline(1919,0,1919,1439);
    lline(0,1439,1919,1439);

    lline(50,720,1870,720);
    lline(960,40,960,1400);

    for (j=0;j<=30;j++) {
        xa[j]=60*(j+1);

```

```
    y=sin((60*(j+1)-960)*fx);
    ya[j]=720-(y/fy);
}

for (j=0;j<=27;j++) {
    mcurve(xa[j],ya[j],xa[j+1],ya[j+1],xa[j+2],ya[j+2],
        xa[j+3],ya[j+3]);
}

GrInit;
PrintPic(pix);

SetCursor(3.8,1.7);
printf("y");
SetCursor(7.2,3.7);
printf("x");
SetCursor(3.7,6.7);
printf("y = sin x");

FormFeed;
Finish;
}
```

We describe how the `scurve` and `mcurve` functions work. The function

```
scurve(a1,b1,p1,q1,p2,q2,a2,b2)
```

draws a curve  $\gamma$  from  $(p_1, q_1)$  to  $(p_2, q_2)$ , in `pix` space. The curve can be defined as follows. Say  $\gamma(t) = (x(t), y(t))$ ,  $0 \leq t \leq 1$ . Then

$$(5.1) \quad x(0) = p_1, \quad x(1) = p_2,$$

and similarly  $y(0) = q_1$ ,  $y(1) = q_2$ . We also specify  $x'(t)$  at  $t = 0, 1$ , by

$$(5.2) \quad x'(0) = \frac{1}{2}(p_2 - a_1) = s_1, \quad x'(1) = \frac{1}{2}(a_2 - p_1) = s_2,$$

and similarly  $y'(0) = \frac{1}{2}(q_2 - b_1)$ ,  $y'(1) = \frac{1}{2}(b_2 - q_1)$ . Then  $x(t)$  is defined to be the unique cubic polynomial

$$(5.3) \quad x(t) = c_0 + c_1t + c_2t^2 + c_3t^3$$

satisfying (5.1)–(5.2). Evaluation at  $t = 0$  yields

$$(5.4) \quad c_0 = p_1, \quad c_1 = s_1,$$

and then evaluation at  $t = 1$  yields

$$(5.5) \quad c_2 = 3(p_2 - p_1) - 2s_1 - s_2, \quad c_3 = 2(p_1 - p_2) + s_1 + s_2.$$

The polynomial  $y(t) = d_0 + d_1t + d_2t^2 + d_3t^3$  is similarly defined. The `scurve` function plots  $\gamma(t)$  at  $t = j/50$ ,  $j = 0, 1, 2, \dots, 50$ , while the `mcurve` function plots  $\gamma(t)$  at  $t = j/500$ ,  $j = 0, 1, 2, \dots, 500$ . The curves produced by this method are sometimes called “Bezier curves.”

The next program uses the ‘Box’ and ‘MBox’ commands. We draw one big box, with upper left `pix` coordinates  $(0, 0)$  and lower right coordinates  $(1919, 2399)$ , and eight little boxes inside, labeled `box 1` through `box 8`.

```
/* boxes.c */

#include <stdio.h>

#define MAXK 2400

#include "print.c"
#include "draw.c"
```

```

main()
{
    int j,k;
    extern char pix[1920][2400];

    for (k=0;k<=2399;k++) {
        for (j=0;j<=1919;j++) pix[j][k]=0;
    }

    Box(0,0,1919,2399);

    for (j=0;j<=1;j++) {
        for (k=0;k<=3;k++)
            MBox(80+960*j,50+600*k,880+960*j,550+600*k);
    }

    GrInit;
    PrintPic(pix);

    for (j=0;j<=1;j++) {
        for (k=0;k<=3;k++) {
            SetCursor(2.2+3.2*j,2.5+2.0*k);
            printf("box %d",1+j+2*k);
        }
    }

    FormFeed;
    Finish;
}

```

One final comment on the workings of "draw.c". In the functions `line` and `curve`, `pix` is declared as an external variable. This declaration must also be made in programs that use "draw.c", and the  $1920 \times N$  array affected by these functions must be called `pix`, unless one edits "draw.c".

## 6. More on the PCL5 programming language

A PCL5 command typically has the following structure:

$${}^E c \ \& \ a \ \# \ L$$

These five characters are called, respectively:

escape character  
 parametrized character  
 group character  
 value field  
 termination character

Such a command is known as an ‘escape sequence’ because it begins with the escape character,  ${}^E c$ . The parametrized character signifies that the command will contain a parameter, and it also defines the type of operation the command performs. Some commands, such as  ${}^E c E$  (reset printer) do not contain a parametrized character. The group character identifies the specific command that the escape sequence performs. Here are some examples of group characters (accompanied by parametrized characters when necessary).

&a change a page format parameter  
 &l formatting command  
 ( font command  
 (s set font parameters  
 ) secondary font command  
 )s set secondary font parameters  
 \*r raster graphics command  
 \*b graphics data command

The value field typically contains a number, either an integer or a decimal quantity. The termination character (typically capitalized) is used to mark the end of an escape sequence and to identify the operation performed.

One can merge escape sequences. Here is an example:

$${}^E c \&l 00 {}^E c (8U {}^E c (s1p10v0s3b5T$$

This breaks down as follows:

$E_c\&\ell0O$	portrait mode (O stands for orientation)
$E_c(8U$	select Roman-8 symbol set
$E_c(s1p10v0s3b5T$	

The last escape sequence in turn is compound, and breaks down as follows:

$E_c(s1P$	font will be proportional
$E_c(s10V$	10 point type size
$E_c(s0S$	upright style
$E_c(s3B$	bold stroke
$E_c(s5T$	Times Roman typeface

Note that in the last compound sequence, there is only one  $E_c(s$ , and only the last termination character is capitalized.

The following is a list of some important escape sequences. A more complete list can be found in Chapter 6 of [CPH], and in Chapter 11 of [BR].

job control

$E_cE$	reset printer
$E_c\&\ell\#X$	set number of copies (e.g., $E_c\&\ell10X$ for ten copies)

page control

$E_c\&\ell\#O$	set page orientation ( $\# = 0$ portrait, $\# = 1$ , landscape)
$E_c\&\ell\#E$	set top margin
$E_c\&\ell\#F$	set text length
$E_c\&a\#L$	set left margin
$E_c\&a\#M$	set right margin
$E_c\&\ell\#D$	set line spacing ( $E_c\&\ell6D$ is 6 lines per inch)
FF	eject page

cursor positioning

$E_c\&a\#R$	move cursor to row $\#$
$E_c\&p\#Y$	set vertical cursor position in dots
$E_c\&a\#V$	set vertical cursor position in decipoints
$E_c\&a\#C$	move cursor to column $\#$

$E_c\&p\#X$  set horizontal cursor position in dots  
 $E_c\&a\#H$  set horizontal cursor position in decipoints  
 $E_c=$  half-line feed  
 BS backspace  
 LF line feed  
 SP space  
 HT horizontal tab  
 $E_c\&k\#G$  line termination  
 $E_c\&f0S$  store current cursor position  
 $E_c\&f1S$  move cursor to last stored position

#### font selection

$E_c(\#$  designate primary symbol set  
 $E_c)\#$  designate secondary symbol set  
 $E_c(s\#P$  designate primary font spacing  
 $E_c)s\#P$  designate secondary font spacing  
 $E_c(s\#H$  designate primary font pitch (e.g.,  $E_c(s0p10H$  for ten char per inch)  
 $E_c)s\#H$  designate secondary font pitch  
 $E_c\&k0S$  set primary and secondary font pitch to 10 cpi  
 $E_c(s\#V$  designate primary font size  
 $E_c(s\#S$  designate primary font style  
 $E_c(s\#B$  designate primary stroke weight  
 $E_c(s\#T$  designate primary typeface

#### graphics

$E_c^*t\#R$  set raster graphics resolution ( $\# = 75, 100, 150, \text{ or } 300$ )  
 $E_c^*r\#A$  start raster graphics mode  
 $E_c^*b\#W$ [data] transfer raster graphics data  
 $E_c^*rB$  exit raster graphics mode  
 $E_c^*r\#T$  set raster height  
 $E_c^*r\#S$  set raster width  
 $E_c^*c\#P$  fill rectangular area (0 = black, 1 = white, 2 = shaded)

## 7. Color graphics

In this section we discuss the use of "colorprint.c" and "colordraw.c", which can be included in programs that produce pictures on a color HP InkJet printer. We begin with a presentation of cmndb.c, which produces the same piece of the Mandelbrot set as done by mndbt.c, discussed in §4, but this time in color. One can see a few differences between the programs. For one, this time we include "colorprint.c", of course. Also, the array pix assumes values other than 0 or 1; in fact it can assume values in [0, ..., 15], different numbers corresponding to different colors. We say more about this after the following program listing.

```

/* cmndb.c */

#include <stdio.h>

#define MAXK 2400

#include "colorprint.c"

#include "colordraw.c"

char pix[1920][2400];

main()
{
    int j,k,m,c;
    extern char pix[1920][2400];
    double a,b,l,x,y,x1,u,v;

    a=-0.7454285;
    b=0.1130087;
    l=0.0000125;

    for (k=0;k<=2399;k++) {
        for (j=0;j<=1919;j++) pix[j][k]=15;
    }

    for (k=0;k<=2399;k++) {
        for (j=0;j<=1919;j++) {
            x=a+l*(1200-k)/960;

```

```
y=b+1*(960-j)/960;
u=x;
v=y;
m=1;
while ((x*x+y*y<=4) & (m<850)) {
    x1=x;
    x=x*x-y*y+u;
    y=2*x1*y+v;
    m=m+1;
}
c=0;
if (m<800) c=4;
if (m<700) c=5;
if (m<600) c=13;
if (m<500) c=7;
if (m<400) c=3;
if (m<320) c=1;
if (m<280) c=10;
if (m<220) c=14;
if (m<185) c=15;
pix[j][k]=c;
}
}

Box(0,0,1919,2399);

GrInit;
PrintPic(pix);

FormFeed;
Finish;
}
```

We now describe how the numerical values of `pix[j][k]` correspond to colors, in the following table. The various colors, black, blue, green, red, magenta, yellow, and white, are listed in the left column. In the next column are values `c` of `pix` that produce these colors. Note that black corresponds to `c = 0` while white corresponds to `c = 15` (also to `c = 7` or `8`, or any value  $> 15$ ). This contrasts with the setup for black and white printing, using "`print.c`", where 0 yields white and 1 (or any nonzero quantity) yields black.

**Color Table**

	<code>c</code>	black	cyan	magenta	yellow
black	0	X			
blue	1		X	X	
green	2		X		X
cyan	3		X		
red	4			X	X
magenta	5			X	
yellow	6				X
white	7				
white	8				
blue	9		X	X	
green	10		X		X
cyan	11		X		
red	12			X	X
magenta	13			X	
yellow	14				X
white	15				

All other `c` values yield white.

We will discuss the meaning of the other columns in this table after presenting the file "`colorprint.c`", which follows.

```

/* colorprint.c */

#define FormFeed printf("%c",12)

#define GrEnds printf("%c*rB",27)

#define GrInit printf("%cE%c&l1H%c&l0%c*p0X%c*p0Y \

```

```

        %c*t300R",27,27,27,27,27,27)

#define Finish printf("%c&10%c(8U%c(sp10h12vsb3T%c&11H", \
        27,27,27,27)
/* End printing with: GrEnds, FormFeed, Finish */

#define SetCursor(a,b) printf("%c&a%6.1fh %6.1fV", \
        27,720*(a)-180,720*(b)-360)
/* Cursor position set at a inches from left edge, */
/* b inches below top edge */

void ColorRow(double a, double b, int p[1920])
{
    int j,k,m,gp;

    SetCursor(a,b);
    printf("%c*r-4U",27);
    printf("%c*r1A",27);

    printf("%c*b%dV",27,240);
    for (j=0;j<=239;j++) {
        m=0;
        for (k=0;k<=7;k++) {
            m=2*m;
            gp=p[8*j+k];
            if (gp==0) m=m+1;
        }
        putchar(m);
    }

    printf("%c*b%dV",27,240);
    for (j=0;j<=239;j++) {
        m=0;
        for (k=0;k<=7;k++) {
            m=2*m;
            gp=p[8*j+k];
            if ((gp==1) || (gp==2) || (gp==3) || (gp==9)
                || (gp==10) || (gp==11)) m=m+1;
        }
        putchar(m);
    }
}

```

```

    }

    printf("%c*b%dV",27,240);
    for (j=0;j<=239;j++) {
        m=0;
        for (k=0;k<=7;k++) {
            m=2*m;
            gp=[8*j+k];
            if ((gp==1) || (gp==4) || (gp==5) || (gp==9)
                || (gp==12) || (gp==13)) m=m+1;
        }
        putchar(m);
    }

    printf("%c*b%dW",27,240);
    for (j=0;j<=239;j++) {
        m=0;
        for (k=0;k<=7;k++) {
            m=2*m;
            gp=p[8*j+k];
            if ((gp==2) || (gp==4) || (gp==6) || (gp==10)
                || (gp==12) || (gp==14)) m=m+1;
        }
        putchar(m);
    }
}

/* draws a line using 1920 data points */
/* line starts a inches from left edge, */
/* b inches below top edge */

void PrintPic(char g[1920][MAXK])
{
    int j,k;
    double ep;
    int dat[1920];

    ep=1.0/300;
    for (k=0;k<=MAXK-1;k++) {
        for (j=0;j<=1919;j++) dat[j]=g[j][k];
        ColorRow(1.0,1.5+k*ep,dat);
    }
}

```

```

    }
}

void RmFont(int k)
{
    printf("%c(8U",27);
    printf("%c(s1p%dv0s0b4101T",27,k);
}

```

We briefly discuss the contents of "colorprint.c". The defined quantities are like those in "print.c", except that we have omitted `#define PutRow`. The ingredients of this are included in the code in `ColorRow`, the analogue here of `BWRow`. One ingredient of the old `PutRow`, `printf("%c*r1A",27)`, appears in `ColorRow`, preceded by `printf("%c*r-4U",27)`, which specifically has to do with setting up color graphics. The other ingredient of `PutRow`, `printf("%c*b%dW",27,240)`, is expanded into four such messages, three of which end with a "V" rather than a "W". We are making four passes to draw a line in color. The first pass determines whether black ink is to be used. The next three passes determine whether cyan, magenta, and/or yellow ink is to be used on each dot. The way the various colors (blue, green, cyan, red, etc.) are made up of these ingredients is specified in the Color Table printed a few pages back.

Next we present the file "colordraw.c". This is very much like "draw.c"; its function is to draw black lines and black curves. The only difference lies in the fact that black now corresponds to  $c = 0$  rather than 1.

```

/* colordraw.c */

/* #define MAXK in your program */

#define lline(p1,q1,p2,q2) line(3500,p1,q1,p2,q2)
#define mline(p1,q1,p2,q2) line(500,p1,q1,p2,q2)
#define sline(p1,q1,p2,q2) line(50,p1,q1,p2,q2)

void line(int length, int p1, int q1, int p2, int q2)
{
    int i,j,k;

```

```

double lng;
extern char pix[1920][MAXK];
lng=1.0*length;
for (i=0;i<=length;i++) {
    j=(1.0*i*p1+(lng-1.0*i)*p2)/lng;
    k=(1.0*i*q1+(lng-1.0*i)*q2)/lng;
    pix[j][k]=0;
}
}

#define mcurve(a1,b1,p1,q1,p2,q2,a2,b2) \
    curve(500,a1,b1,p1,q1,p2,q2,a2,b2)

#define scurve(a1,b1,p1,q1,p2,q2,a2,b2) \
    curve(50,a1,b1,p1,q1,p2,q2,a2,b2)

void curve(int length, int a1, int b1, int p1, int q1,
           int p2, int q2, int a2, int b2)
{
    int i,j,k;
    double lng,s1,s2,c0,c1,c2,c3,d0,d1,d2,d3,t;
    extern char pix[1920][MAXK];

    lng=1.0*length;
    s1=0.5*(p2-a1);
    s2=0.5*(a2-p1);
    c0=1.0*p1;
    c1=s1;
    c2=3.0*(p2-p1)-2*s1-s2;
    c3=2.0*(p1-p2)+s1+s2;

    s1=0.5*(q2-b1);
    s2=0.5*(b2-q1);
    d0=1.0*q1;
    d1=s1;
    d2=3.0*(q2-q1)-2*s1-s2;
    d3=2.0*(q1-q2)+s1+s2;

    for (i=0;i<=length;i++) {
        t=i/lng;
        j=c0+t*(c1+t*(c2+t*c3));
        k=d0+t*(d1+t*(d2+t*d3));
    }
}

```

```
        pix[j][k]=0;
    }
}

void Box(int p1, int q1, int p2, int q2)
{
    lline(p1,q1,,p2,q1);
    lline(p1,q1,p1,q2);
    lline(p2,q1,p2,q2);
    lline(p1,q2,p2,q2);
}

void MBox(int p1, int q1, int p2, int q2)
{
    mline(p1,q1,p2,q1);
    mline(p1,q1,p1,q2);
    mline(p2,q1,p2,q2);
    mline(p1,q2,p2,q2);
}
```

## References

- [BR] S. Bennett and P. Randall, *The LaserJet III Companion*, Brady Publ., New York, 1991.
- [CPH] M. Crane, J. Pierce, and D. Holzgag, *LaserJet Companion*, Microsoft Press, 1991.
- [Ez] B. Ezzell, *Graphics Programming in Turbo Pascal*, Addison-Wesley, New York, 1990.
- [L] E. de Castro Lopo, et al., *C for Linux Programming*, SAMS, McMillan, Indianapolis, In., 2000.
- [SS] R. Smith and C. Stevenson, *LaserJet 4, Typography and Graphics*, Randon House, New York, 1993.
- [T] M. Taylor, *Graphics on the LaserJet III via Turbo Pascal*, Lecture Notes, 1994.