มหาวิทยาลัยสงขลานครินทร์
คณะวิศวกรรมศาสตร์

สอบกลางภาค: ภาคการศึกษาที่ 1 ปีการศึกษา: 2550

วันที่สอบ: 3 สิงหาคม 2550 เวลา: 0900-1200

วิชา: 240-620 Advanced Unix Network Programming ห้อง: A401

คำสั่ง:

อนุญาตให้นำหนังสือหรือเอกสารเข้าห้องสอบได้

ข้อสอบมีทั้งหมด 24 ข้อ คะแนนรวม 50 คะแนน ให้ทำทุกข้อ

Compiler (4 marks)

1. What is an option for gcc to compile a C source code to an object code?
2. What is an option for gcc to compile a C source code to a binary program, assume that no other object except system library are needed to build that binary?
3. What is an option for gcc to compile a C source code to a statical linked binary?
4. Without giving specific output name for compiling with gcc, what is the binary name that gcc will create?

Debugger (5 marks)

5. What is source-level debugging?
6. What need to be done with gcc to allow source-level debugging using gdb?
7. What is a gdb command that use to print source code for a specific functions?
8. What is a gdb command that use to set breakpoint for at a specific line.
9. How can command line arguments of the program supply to the program when being debugged by gdb?

Creating Library (5 marks)

10. What is the program use to create a static library from objects file?
11. If there are 4 object files t1.o t2.o t3.o t4.o to be created as library libt.a, what is the command line use to create that library?
12. If a program main.c requires functions from libt.a in previous question how can it linked to output binary main?
13. What is compiler options required to create objects file that will be used to created shared library?
14. What is the command that use for list symbols in library or binary program?

Using Make (10 marks)

15. An application that you are going to develop depend on 3 libraries libt.a libx.a and math library (system library). Library libt.a will be create from 3 functions t1(), t2(), t3() with stored in separate 3 files t1.c t2.c t3.c respectively. There is t.h header file that is used by all these functions. Library libx.a also will created from 3 functions x1(), x2(), x3(), which also separate to 3 files x1.c x2.c x3.c. x1() and x2() required definitions defined in x.h header file. x3() doesn't required that. The main function will be stored in main.c which include x.h and t.h. Using this information to **create a makefile** for making this application, Makefile should contains definition for CC, CFLAGS and other macros (if needed). It should also has clean target for removing all objects, binary and library files leaving only source file. The final target binary should be called myApp, and don't forget that it also required math library, apart of libt.a and libx.a.

Programming Conventions using GNU/Linux (5 marks)

16. What main() function prototype to be used if environment (such as HOME, SHELL) are needed?
17. Passing command line options to program using getopt_long function, what is the meaning of ":" in short options.
18. Write long options structure equivalent to this short options "a:b:cde" that can be used to provide to getopt_long() function.
19. Which function should be used to create a temporary file, if standard I/O library (fopen, fread, ..., etc) are used to operate with that file.
20. How can a program set exit code that can be check by shell using "$?" variable?

Process (6 marks)

Write a short program to execute command "ps aux" using

21. system() system call
22. execlp() system call
23. execvp() system call

Program must be complete (has necessary headers and all variables), functions can be called in main().

Programming (15 marks)

24. Using the following source code of bmpviewer program `bmpview.c`, `bitmap.c`, `bitmap.h` as example to write histogram.c a program to count a number of pixel for each color level in RED, GREEN, BLUE component of a pixel. You can call functions from those example program, if needed. You can use any method to display the histogram.

Chatchai J

2007-08-01

bitmap.h

```
/*
 * Windows BMP file definitions for OpenGL.
 *
 * Written by Michael Sweet.
 */

#ifndef _BITMAP_H_
#  define _BITMAP_H_

/*
 * Include necessary headers.
 */

#  include <GL/glut.h>
#  ifdef WIN32
#    include <windows.h>
#    include <wingdi.h>
#  endif /* WIN32 */

/*
 * Make this header file work with C and C++ source code...
 */

#  ifdef __cplusplus
extern "C" {
#  endif /* __cplusplus */


/*
 * Bitmap file data structures (these are defined in <wingdi.h> under
 * Windows...)
 *
 * Note that most Windows compilers will pack the following structures, so
 * when reading them under MacOS or UNIX we need to read individual fields
 * to avoid differences in alignment...
 */

#  ifndef WIN32
typedef struct                          /**** BMP file header structure ****/
    {
    unsigned short bfType;              /* Magic number for file */
    unsigned int   bfSize;              /* Size of file */
    unsigned short bfReserved1;         /* Reserved */
    unsigned short bfReserved2;         /* ... */
    unsigned int   bfOffBits;           /* Offset to bitmap data */
    } BITMAPFILEHEADER;

#  define BF_TYPE 0x4D42                /* "MB" */

typedef struct                          /**** BMP file info structure ****/
    {
    unsigned int   biSize;              /* Size of info header */
    int            biWidth;             /* Width of image */
    int            biHeight;            /* Height of image */
    unsigned short biPlanes;            /* Number of color planes */
    unsigned short biBitCount;          /* Number of bits per pixel */
    unsigned int   biCompression;       /* Type of compression to use */
    unsigned int   biSizeImage;         /* Size of image data */
    int            biXPelsPerMeter;     /* X pixels per meter */
    int            biYPelsPerMeter;     /* Y pixels per meter */
    unsigned int   biClrUsed;           /* Number of colors used */
    unsigned int   biClrImportant;      /* Number of important colors */
    } BITMAPINFOHEADER;

/*
 * Constants for the biCompression field...
 */

#  define BI_RGB       0                /* No compression - straight BGR data */
#  define BI_RLE8      1                /* 8-bit run-length compression */
#  define BI_RLE4      2                /* 4-bit run-length compression */
#  define BI_BITFIELDS 3                /* RGB bitmap with RGB masks */

typedef struct                          /**** Colormap entry structure ****/
    {
    unsigned char  rgbBlue;             /* Blue value */
    unsigned char  rgbGreen;            /* Green value */
    unsigned char  rgbRed;              /* Red value */
    unsigned char  rgbReserved;         /* Reserved */
    } RGBQUAD;

typedef struct                          /**** Bitmap information structure ****/
    {
    BITMAPINFOHEADER bmiHeader;         /* Image header */
    RGBQUAD          bmiColors[256];    /* Image colormap */
    } BITMAPINFO;
#  endif /* !WIN32 */

/*
 * Prototypes...
 */
```

```
extern GLubyte *LoadDIBitmap(const char *filename, BITMAPINFO **info);
extern int     SaveDIBitmap(const char *filename, BITMAPINFO *info,
                            GLubyte *bits);

#  ifdef __cplusplus
}
#  endif /* __cplusplus */
#endif /* !_BITMAP_H_ */
```

bitmap.c

```
/*
 * Windows BMP file functions for OpenGL.
 *
 * Written by Michael Sweet.
 */

#include "bitmap.h"
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>


#ifdef WIN32
/*
 * 'LoadDIBitmap()' - Load a DIB/BMP file from disk.
 *
 * Returns a pointer to the bitmap if successful, NULL otherwise...
 */

GLubyte *                       /* O - Bitmap data */
LoadDIBitmap(const char *filename, /* I - File to load */
             BITMAPINFO **info)    /* O - Bitmap information */
    {
    FILE              *fp;          /* Open file pointer */
    GLubyte           *bits;        /* Bitmap pixel bits */
    int               bitsize;      /* Size of bitmap */
    int               infosize;     /* Size of header information */
    BITMAPFILEHEADER header;        /* File header */


    /* Try opening the file; use "rb" mode to read this *binary* file. */
    if ((fp = fopen(filename, "rb")) == NULL)
        return (NULL);

    /* Read the file header and any following bitmap information... */
    if (fread(&header, sizeof(BITMAPFILEHEADER), 1, fp) < 1)
        {
        /* Couldn't read the file header - return NULL... */
          fclose(fp);
        return (NULL);
        }

    if (header.bfType != 'MB')          /* Check for BM reversed... */
        {
        /* Not a bitmap file - return NULL... */
        fclose(fp);
        return (NULL);
        }

    infosize = header.bfOffBits - sizeof(BITMAPFILEHEADER);
    if ((*info = (BITMAPINFO *)malloc(infosize)) == NULL)
        {
        /* Couldn't allocate memory for bitmap info - return NULL... */
        fclose(fp);
        return (NULL);
        }

    if (fread(*info, 1, infosize, fp) < infosize)
        {
        /* Couldn't read the bitmap header - return NULL... */
        free(*info);
        fclose(fp);
        return (NULL);
        }

    /* Now that we have all the header info read in, allocate memory for *
     * the bitmap and read *it* in...                                    */
    if ((bitsize = (*info)->bmiHeader.biSizeImage) == 0)
        bitsize = ((*info)->bmiHeader.biWidth *
                   (*info)->bmiHeader.biBitCount + 7) / 8 *
                     abs((*info)->bmiHeader.biHeight);

    if ((bits = malloc(bitsize)) == NULL)
        {
        /* Couldn't allocate memory - return NULL! */
        free(*info);
        fclose(fp);
        return (NULL);
```

```
    }

  if (fread(bits, 1, bitsize, fp) < bitsize)
    {
    /* Couldn't read bitmap - free memory and return NULL! */
    free(*info);
    free(bits);
    fclose(fp);
    return (NULL);
    }

  /* OK, everything went fine - return the allocated bitmap... */
  fclose(fp);
  return (bits);
  }


/*
 * 'SaveDIBitmap()' - Save a DIB/BMP file to disk.
 *
 * Returns 0 on success or -1 on failure...
 */

int                                /* O - 0 = success, -1 = failure */
SaveDIBitmap(const char *filename, /* I - File to load */
             BITMAPINFO *info,     /* I - Bitmap information */
               GLubyte    *bits)     /* I - Bitmap data */
  {
  FILE              *fp;          /* Open file pointer */
  int               size,         /* Size of file */
                    infosize,     /* Size of bitmap info */
                    bitsize;      /* Size of bitmap pixels */
  BITMAPFILEHEADER header;        /* File header */


  /* Try opening the file; use "wb" mode to write this *binary* file. */
  if ((fp = fopen(filename, "wb")) == NULL)
    return (-1);

  /* Figure out the bitmap size */
  if (info->bmiHeader.biSizeImage == 0)
    bitsize = (info->bmiHeader.biWidth *
       info->bmiHeader.biBitCount + 7) / 8 *
                abs(info->bmiHeader.biHeight);
  else
    bitsize = info->bmiHeader.biSizeImage;

  /* Figure out the header size */
  infosize = sizeof(BITMAPINFOHEADER);
  switch (info->bmiHeader.biCompression)
    {
    case BI_BITFIELDS :
      infosize += 12; /* Add 3 RGB doubleword masks */
      if (info->bmiHeader.biClrUsed == 0)
        break;
    case BI_RGB :
      if (info->bmiHeader.biBitCount > 8 &&
      info->bmiHeader.biClrUsed == 0)
        break;
    case BI_RLE8 :
    case BI_RLE4 :
      if (info->bmiHeader.biClrUsed == 0)
        infosize += (1 << info->bmiHeader.biBitCount) * 4;
        else
        infosize += info->bmiHeader.biClrUsed * 4;
        break;
    }

  size = sizeof(BITMAPFILEHEADER) + infosize + bitsize;

  /* Write the file header, bitmap information, and bitmap pixel data... */
  header.bfType      = 'MB'; /* Non-portable... sigh */
  header.bfSize      = size;
  header.bfReserved1 = 0;
  header.bfReserved2 = 0;
  header.bfOffBits   = sizeof(BITMAPFILEHEADER) + infosize;

  if (fwrite(&header, 1, sizeof(BITMAPFILEHEADER), fp) < sizeof(BITMAPFILEHEADER))
    {
    /* Couldn't write the file header - return... */
    fclose(fp);
    return (-1);
    }

  if (fwrite(info, 1, infosize, fp) < infosize)
    {
    /* Couldn't write the bitmap header - return... */
    fclose(fp);
    return (-1);
    }

  if (fwrite(bits, 1, bitsize, fp) < bitsize)
    {
    /* Couldn't write the bitmap - return... */
```

```
        fclose(fp);
        return (-1);
        }

    /* OK, everything went fine - return... */
    fclose(fp);
    return (0);
    }


#else /* !WIN32 */
/*
 * Functions for reading and writing 16- and 32-bit little-endian integers.
 */

static unsigned short read_word(FILE *fp);
static unsigned int   read_dword(FILE *fp);
static int            read_long(FILE *fp);

static int            write_word(FILE *fp, unsigned short w);
static int            write_dword(FILE *fp, unsigned int dw);
static int            write_long(FILE *fp, int l);


/*
 * 'LoadDIBitmap()' - Load a DIB/BMP file from disk.
 *
 * Returns a pointer to the bitmap if successful, NULL otherwise...
 */

GLubyte *                       /* O - Bitmap data */
LoadDIBitmap(const char *filename, /* I - File to load */
             BITMAPINFO **info)    /* O - Bitmap information */
    {
    FILE             *fp;          /* Open file pointer */
    GLubyte          *bits;        /* Bitmap pixel bits */
    GLubyte          *ptr;         /* Pointer into bitmap */
    GLubyte          temp;         /* Temporary variable to swap red and blue */
    int              x, y;         /* X and Y position in image */
    int              length;       /* Line length */
    int              bitsize;      /* Size of bitmap */
    int              infosize;     /* Size of header information */
    BITMAPFILEHEADER header;       /* File header */


    /* Try opening the file; use "rb" mode to read this *binary* file. */
    if ((fp = fopen(filename, "rb")) == NULL)
        return (NULL);

    /* Read the file header and any following bitmap information... */
    header.bfType      = read_word(fp);
    header.bfSize      = read_dword(fp);
    header.bfReserved1 = read_word(fp);
    header.bfReserved2 = read_word(fp);
    header.bfOffBits   = read_dword(fp);

    if (header.bfType != BF_TYPE) /* Check for BM reversed... */
        {
        /* Not a bitmap file - return NULL... */
        fclose(fp);
        return (NULL);
        }

    infosize = header.bfOffBits - 18;
    if ((*info = (BITMAPINFO *)malloc(sizeof(BITMAPINFO))) == NULL)
        {
        /* Couldn't allocate memory for bitmap info - return NULL... */
        fclose(fp);
        return (NULL);
        }

    (*info)->bmiHeader.biSize          = read_dword(fp);
    (*info)->bmiHeader.biWidth         = read_long(fp);
    (*info)->bmiHeader.biHeight        = read_long(fp);
    (*info)->bmiHeader.biPlanes        = read_word(fp);
    (*info)->bmiHeader.biBitCount      = read_word(fp);
    (*info)->bmiHeader.biCompression   = read_dword(fp);
    (*info)->bmiHeader.biSizeImage     = read_dword(fp);
    (*info)->bmiHeader.biXPelsPerMeter = read_long(fp);
    (*info)->bmiHeader.biYPelsPerMeter = read_long(fp);
    (*info)->bmiHeader.biClrUsed       = read_dword(fp);
    (*info)->bmiHeader.biClrImportant  = read_dword(fp);

    if (infosize > 40)
          if (fread((*info)->bmiColors, infosize - 40, 1, fp) < 1)
            {
            /* Couldn't read the bitmap header - return NULL... */
            free(*info);
            fclose(fp);
            return (NULL);
            }

    /* Now that we have all the header info read in, allocate memory for *
     * the bitmap and read *it* in...                                    */
```

```
    if ((bitsize = (*info)->bmiHeader.biSizeImage) == 0)
        bitsize = ((*info)->bmiHeader.biWidth *
                   (*info)->bmiHeader.biBitCount + 7) / 8 *
                    abs((*info)->bmiHeader.biHeight);

    if ((bits = malloc(bitsize)) == NULL)
        {
        /* Couldn't allocate memory - return NULL! */
        free(*info);
        fclose(fp);
        return (NULL);
        }

    if (fread(bits, 1, bitsize, fp) < bitsize)
        {
        /* Couldn't read bitmap - free memory and return NULL! */
        free(*info);
        free(bits);
        fclose(fp);
        return (NULL);
        }

    /* Swap red and blue */
    length = ((*info)->bmiHeader.biWidth * 3 + 3) & ~3;
    for (y = 0; y < (*info)->bmiHeader.biHeight; y ++)
        for (ptr = bits + y * length, x = (*info)->bmiHeader.biWidth;
             x > 0;
             x --, ptr += 3)
            {
            temp   = ptr[ 0] ;
            ptr[ 0]  = ptr[ 2] ;
            ptr[ 2]  = temp;
            }

    /* OK, everything went fine - return the allocated bitmap... */
    fclose(fp);
    return (bits);
    }


/*
 * 'SaveDIBitmap()' - Save a DIB/BMP file to disk.
 *
 * Returns 0 on success or -1 on failure...
 */

int                                     /* O - 0 = success, -1 = failure */
SaveDIBitmap(const char *filename,      /* I - File to load */
             BITMAPINFO *info,          /* I - Bitmap information */
               GLubyte    *bits)        /* I - Bitmap data */
    {
    FILE *fp;                           /* Open file pointer */
    int  size,                          /* Size of file */
         infosize,                      /* Size of bitmap info */
         bitsize;                       /* Size of bitmap pixels */


    /* Try opening the file; use "wb" mode to write this *binary* file. */
    if ((fp = fopen(filename, "wb")) == NULL)
        return (-1);

    /* Figure out the bitmap size */
    if (info->bmiHeader.biSizeImage == 0)
          bitsize = (info->bmiHeader.biWidth *
             info->bmiHeader.biBitCount + 7) / 8 *
                     abs(info->bmiHeader.biHeight);
    else
          bitsize = info->bmiHeader.biSizeImage;

    /* Figure out the header size */
    infosize = sizeof(BITMAPINFOHEADER);
    switch (info->bmiHeader.biCompression)
          {
          case BI_BITFIELDS :
            infosize += 12; /* Add 3 RGB doubleword masks */
            if (info->bmiHeader.biClrUsed == 0)
                break;
          case BI_RGB :
            if (info->bmiHeader.biBitCount > 8 &&
          info->bmiHeader.biClrUsed == 0)
                break;
          case BI_RLE8 :
          case BI_RLE4 :
            if (info->bmiHeader.biClrUsed == 0)
              infosize += (1 << info->bmiHeader.biBitCount) * 4;
              else
              infosize += info->bmiHeader.biClrUsed * 4;
              break;
          }

    size = sizeof(BITMAPFILEHEADER) + infosize + bitsize;

    /* Write the file header, bitmap information, and bitmap pixel data... */
    write_word(fp, BF_TYPE);        /* bfType */
```

```
    write_dword(fp, size);          /* bfSize */
    write_word(fp, 0);              /* bfReserved1 */
    write_word(fp, 0);              /* bfReserved2 */
    write_dword(fp, 18 + infosize); /* bfOffBits */

    write_dword(fp, info->bmiHeader.biSize);
    write_long(fp, info->bmiHeader.biWidth);
    write_long(fp, info->bmiHeader.biHeight);
    write_word(fp, info->bmiHeader.biPlanes);
    write_word(fp, info->bmiHeader.biBitCount);
    write_dword(fp, info->bmiHeader.biCompression);
    write_dword(fp, info->bmiHeader.biSizeImage);
    write_long(fp, info->bmiHeader.biXPelsPerMeter);
    write_long(fp, info->bmiHeader.biYPelsPerMeter);
    write_dword(fp, info->bmiHeader.biClrUsed);
    write_dword(fp, info->bmiHeader.biClrImportant);

    if (infosize > 40)
          if (fwrite(info->bmiColors, infosize - 40, 1, fp) < 1)
            {
            /* Couldn't write the bitmap header - return... */
            fclose(fp);
            return (-1);
            }

    if (fwrite(bits, 1, bitsize, fp) < bitsize)
        {
        /* Couldn't write the bitmap - return... */
        fclose(fp);
        return (-1);
        }

    /* OK, everything went fine - return... */
    fclose(fp);
    return (0);
    }


/*
 * 'read_word()' - Read a 16-bit unsigned integer.
 */

static unsigned short     /* O - 16-bit unsigned integer */
read_word(FILE *fp)       /* I - File to read from */
    {
    unsigned char b0, b1; /* Bytes from file */

    b0 = getc(fp);
    b1 = getc(fp);

    return ((b1 << 8) | b0);
    }


/*
 * 'read_dword()' - Read a 32-bit unsigned integer.
 */

static unsigned int               /* O - 32-bit unsigned integer */
read_dword(FILE *fp)              /* I - File to read from */
    {
    unsigned char b0, b1, b2, b3; /* Bytes from file */

    b0 = getc(fp);
    b1 = getc(fp);
    b2 = getc(fp);
    b3 = getc(fp);

    return ((((((b3 << 8) | b2) << 8) | b1) << 8) | b0);
    }


/*
 * 'read_long()' - Read a 32-bit signed integer.
 */

static int                        /* O - 32-bit signed integer */
read_long(FILE *fp)               /* I - File to read from */
    {
    unsigned char b0, b1, b2, b3; /* Bytes from file */

    b0 = getc(fp);
    b1 = getc(fp);
    b2 = getc(fp);
    b3 = getc(fp);

    return ((int)(((((b3 << 8) | b2) << 8) | b1) << 8) | b0);
    }


/*
 * 'write_word()' - Write a 16-bit unsigned integer.
 */
```

```
static int                      /* O - 0 on success, -1 on error */
write_word(FILE           *fp, /* I - File to write to */
           unsigned short w)    /* I - Integer to write */
    {
    putc(w, fp);
    return (putc(w >> 8, fp));
    }


/*
 * 'write_dword()' - Write a 32-bit unsigned integer.
 */

static int                      /* O - 0 on success, -1 on error */
write_dword(FILE         *fp, /* I - File to write to */
            unsigned int dw)  /* I - Integer to write */
    {
    putc(dw, fp);
    putc(dw >> 8, fp);
    putc(dw >> 16, fp);
    return (putc(dw >> 24, fp));
    }


/*
 * 'write_long()' - Write a 32-bit signed integer.
 */

static int           /* O - 0 on success, -1 on error */
write_long(FILE *fp, /* I - File to write to */
           int  l)   /* I - Integer to write */
    {
    putc(l, fp);
    putc(l >> 8, fp);
    putc(l >> 16, fp);
    return (putc(l >> 24, fp));
    }
#endif /* WIN32 */
```

bmpview.c

```
/*
 * OpenGL bitmap viewing demo from Chapter 7.
 *
 * Written by Michael Sweet.
 */

/*
 * Include necessary headers.
 */

#include "bitmap.h"
#include <GL/glut.h>


/*
 * Globals...
 */

int        Width;       /* Width of window */
int        Height;      /* Height of window */
BITMAPINFO *BitmapInfo; /* Bitmap information */
GLubyte    *BitmapBits; /* Bitmap data */


/*
 * Functions...
 */

void Redraw(void);
void Resize(int width, int height);


/*
 * 'main()' - Open a window and display a bitmap.
 */

int                /* O - Exit status */
main(int  argc,    /* I - Number of command-line arguments */
     char *argv[]) /* I - Command-line arguments */
    {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowSize(792, 573);
    glutCreateWindow("Bitmap File Viewer");
    glutReshapeFunc(Resize);
    glutDisplayFunc(Redraw);

    if (argc > 1)
        BitmapBits = LoadDIBitmap(argv[1], &BitmapInfo);
    else
```

```
        BitmapBits = LoadDIBitmap("mountain.bmp", &BitmapInfo);

    glutMainLoop();
    if (BitmapInfo)
        {
          free(BitmapInfo);
          free(BitmapBits);
        }
    return (0);
    }


/*
 * 'Redraw()' - Redraw the window...
 */

void
Redraw(void)
    {
    GLfloat xsize, ysize;     /* Size of image */
    GLfloat xoffset, yoffset; /* Offset of image */
    GLfloat xscale, yscale;   /* Scaling of image */

    /* Clear the window to black */
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    if (BitmapInfo)
        {
        xsize = Width;
        ysize = BitmapInfo->bmiHeader.biHeight * xsize /
                BitmapInfo->bmiHeader.biWidth;
        if (ysize > Height)
            {
            ysize = Height;
            xsize = BitmapInfo->bmiHeader.biWidth * ysize /
                    BitmapInfo->bmiHeader.biHeight;
            }

        xscale  = xsize / BitmapInfo->bmiHeader.biWidth;
        yscale  = ysize / BitmapInfo->bmiHeader.biHeight;

        xoffset = (Width - xsize) * 0.5;
        yoffset = (Height - ysize) * 0.5;

        glRasterPos2f(xoffset, yoffset);
        glPixelZoom(xscale, yscale);

        glDrawPixels(BitmapInfo->bmiHeader.biWidth,
                     BitmapInfo->bmiHeader.biHeight,
                     GL_BGR_EXT, GL_UNSIGNED_BYTE, BitmapBits);
        }

    glFinish();
    }


/*
 * 'Resize()' - Resize the window...
 */

void
Resize(int width,  /* I - Width of window */
       int height) /* I - Height of window */
    {
    /* Save the new width and height */
    Width  = width;
    Height = height;

    /* Reset the viewport... */
    glViewport(0, 0, width, height);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, (GLfloat)width, 0.0, (GLfloat)height, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    }
```