CALIFORNIA STATE UNIVERSITY, LONG BEACH

College of Engineering
Department of Computer Engineering and Computer Science
Dr. Thinh V. Nguyen
CECS-553/653: Machine Vision

LAB/ PROJECTS

- **Lab Objectives**: The lab is used mainly to allow students to work on projects and for project discussions. However, it may be necessary that students spend most of their time outside of class.

- **Computer Language and Operating System:** Students are free to use any convenient operating systems and computer languages. However, for purpose of class discussions, Windows OS and C language will be used for compatibility with existing image functions and/or programs. Since the projects are used to supplement the lecture materials, students are expected to use projects to further understanding of the materials. In addition, students are expected to demonstrate the project using the department facilities or their own notebook or laptop computers.

- **Format of project report:** The project report should be typewritten. Use font size of 10-point, font type: Times New Roman, Arial, or Palatino. Single spaced. One and a half spacing between paragraphs. Use headings as appropriate. The entire report should be firmly stapled. Use the project assignment sheet as the cover page. No folder is used. The length of the report depends on the complexity of the project, and may range from 2 pages to 10 pages. *Do not attach program listing*. The report consists of the following sections:

  - *Project Description:* A brief description of the problem statement. Discuss any relevant assumptions if necessary.
  - *Project Background:* Provide a brief summary of background or theory of the techniques used to implement the project.
  - *Algorithm Description*: Discuss the specific algorithms or techniques used to implement the project. Pseudo code or flowcharts should be used where relevant.
  - *Results and Analysis*: Provide the results of the work. Analyze the results. Determine if the results are as expected. Discuss. The results should include outputs of the program (e.g., processed images, tables, useful data).
  - *Conclusion:* Provide a brief conclusion of the project.

- **Project Demonstration:** Students may be required to demonstrate the project. Although this is not a programming class, it is expected that students follow good software engineering practice. In addition, the code should also be efficient (e.g., fast).

- **Image and function libraries:** Please maintain an image database to be used in the projects. Since project work is a continuing effort, it may be necessary for you to use developed image functions again. Therefore, it is useful to maintain these image functions in appropriate library. The images to be used in this class will be bitmap BMP format. Any other format may be used as long as you have a way to read and display the image. This document includes a program using BMP read and write functions to obtain BMP images and to write an image in BMP format. The BMP images can be viewed by any suitable image viewer.

SAMPLE PROGRAM WITH BMP READ AND WRITE FUNCTIONS

Please copy the program and study it carefully. Compile and run the program with the images provided. You of course can also use any images of your own. The program is intended for illustration. It may still contain bugs.

```
// PROGRAM NAME:      bmp_image.cpp
//
// DESCRIPTION:
//
// The objectives of this program include:
//                    (1) to illustrate the BMP image format;
//                    (2) to show how to read and write a BMP image file;
//                    (3) to demonstrate a simple image operation: the local avaraging.
// This program uses the bmp_io.c program posted on the Internet.
// To use this program, an input BMP image file xxx.bmp (where xxx is the
// file name)must be available.
// The program performs the following operations:
//                    (1) read the image file and extract the size information and number
//                              of bits/ pixel.
//                    (2) save the grey level image in the array[][] array.
//                    (3) write the grey level image to the file xxx_grey.bmp.
//                    (4) perform local averaging.
//                    (5) write the averaged image to the file xxx_avg.bmp.
// The bmp_io.c program only supports 24-bit color and 8-bit grey level images.
// The dimensions of the array[][] is 1000x1000.
//
// No attempt to optimize the code is done.  Memory should be de-allocated or
// freed after use.
//
// There should be a view program available to view the .bmp files.

#include "stdafx.h"
// Note that the "stdafx.h" contains the corrected "bmp_io.h"
#include <stdio.h>
#include <stdlib.h>


#define TRUE 1
#define MAX_D 1000


int byte_swap = TRUE;

int main ( long argc, char **argv );

/*****************************************************************************/

int main ( long argc, char **argv ) {

/*****************************************************************************/

  int result, xd, yd, bit, i, j, WSIZE, half, half1,sqsize, min;
  unsigned char array[MAX_D][MAX_D];
  unsigned char *red;
  unsigned char *green;
```

```
unsigned char *blue;
char filename[20],inputfile[20],out1file[20], out2file[20];

printf ("\nThis program reads in an image having BMP format.");
printf ("\nThe program then generates a grey level image and computes the local average.");
printf ("\nNote that the dimensions of the image should be less than 1000x1000\n");
printf ("\nEnter image file name (< 20 characters and no file extension): ");
scanf ("%s",&filename);
strcpy(inputfile, filename);
strcpy(out1file, filename);
strcpy(out2file, filename);
strcat(inputfile,".bmp");
strcat(out1file,"_grey.bmp");
strcat(out2file, "_avg.bmp");


/* Read image file */
result = bmp_read(inputfile, &xd, &yd, &bit, &red, &green, &blue);


if ( result == ERROR ) {
  printf ( "\n" );
  printf ( "BMP_READ failed.\n" );
  return EXIT_FAILURE;
}

printf ("\n--- X Y dimensions = %5d %5d", xd, yd);
printf ("\n--- Number of bits = %5d\n", bit);
if ((xd > MAX_D) || (yd > MAX_D)){
            printf ("\nImage size is too large !");
            return EXIT_FAILURE;
}

if (xd < yd) min = xd;
else min = yd;
do
{
   /* Make sure the window size is not too large, limit to less than
              1/8 of the smaller dimension.
            */
            printf ("\nEnter average window size (odd number, e.g., 5, 7): ");
            scanf ("%d", &WSIZE);
            if (WSIZE > min/8)
                        printf ("\nLocal window is too large !");
            if (!(WSIZE%2))
                        printf ("\nLocal window size must be an odd number !");
}
while ((WSIZE > min/8) || !(WSIZE%2));
sqsize = WSIZE*WSIZE;
half = (WSIZE-1)/2; half1 = half-1;

unsigned char *indexb;
unsigned char *indexg;
unsigned char *indexr;

indexr = red;
```

```
if (bit == 24){
        indexg = green;
        indexb = blue;
}

/*
Copy the input image into the array[][], use only the red plane for grey
level
*/

for (j = 0; j < yd; j++ ) {
  for ( i = 0; i < xd; i++ ) {
                array[i][j] = *indexr;
                *indexg = 0; *indexb = 0;
                indexr = indexr + 1;
                if (bit == 24) {
                        indexg = indexg + 1;
                        indexb = indexb + 1;
                }
  }
}

/*
Write grey level image to xxx_grey.bmp
*/
result = bmp_write (out1file, xd, yd, 8, red, green, blue );
if ( result == ERROR ) {
  printf ( "\n" );
  printf ( "  BMP_WRITE failed.\n" );
  return ERROR;
}
printf ("\nThe grey level image is written to the %s file.",out1file);


/*
Now compute the local average of the grey level image
*/
int sum, k, m;
indexr = red;
indexg = green;
indexb = blue;

/* This operation is local averaging
Each original pixel is replaced by its local average
The effect is to smooth out an image.
If the image is not noisy, the result is a blurry image.
The local averaging has an effect of a low-pass filter to pass
mainly the low frequency components
*/
for (j = 0; j <yd; j++){
        for (i=0; i <xd; i++){
                /* Starting from (0,0) to (xd,yd) to keep the correct
                incrementing of pointers, but process image from
                y = [half, yd-half] and x = [half, xd-half]
                */
                if ((j>half1) && (j<yd-half) && (i>half1) && (i<xd-half1)){
```

```
                                sum = 0;
                                /* At each pixel, open a local window of size WSIZE */
                                 for (k =j-half; k <j+half+1; k++){
                                  for (m = i-half; m <i+half+1; m++){
                                            sum += array[m][k];
                                  }
                                 }

                        *indexr = sum/sqsize;
                        }
                        indexr = indexr + 1;
                        indexg = indexg + 1;
                        indexb = indexb + 1;
                }
  }

/*
  Write the data to a file.
*/
  result = bmp_write (out2file, xd, yd, 8, red, green, blue );

  if ( result == ERROR ) {
    printf ( "\n" );
    printf ( "  BMP_WRITE failed.\n" );
    return ERROR;
  }
  printf ("\nThe averaged image is written to the %s file.",out2file);
  printf ("\n");

  return EXIT_SUCCESS;
}

  /******************************************************************/
  /*  The following are functions available from the Web at the
                        following links:
                        http://www-cse.ucsd.edu/classes/sp03/cse190-b/hw1/bmp_io.c
                        http://orion.math.iastate.edu/burkardt/g_src/bmp_io/bmp_io.h
                        Note that the bmp_io.h in the above link is not correct (some
                        numbers and types of the arguments are wrong).  Compare those
                        with the source code in the following.  The corrected bmp_io.h
                        is contained in the "stdafx.h"
  */
/*************************************************************************************/

int bmp_read ( char *filein_name, int *xsize, int *ysize, int* bsize,
                                unsigned char **rarray, unsigned char **garray, unsigned char **barray )   {

/*************************************************************************************/

/*
  Purpose:

    BMP_READ reads the header and data of a BMP file.

  Modified:
```

```
  05 October 1998
        Apr 05 2003

  Author:

  John Burkardt
        Diem Vu

  Parameters:

  Input, char *FILEIN_NAME, the name of the input file.

  Output, int *XSIZE, *YSIZE, the X and Y dimensions of the image.

        Output, int *bsize, the number of bits per pixel. Only support 8 and 24.

  Output, unsigned char **RARRAY, **GARRAY, **BARRAY, pointers to the red, green
  and blue color arrays.

  Remarks:
  Support 8 bit grayscale and 24 bpp formats only.
*/
  FILE *filein;
  int   numbytes;
  int   psize;
  int   result;
/*
  Open the input file.
*/
  filein = fopen ( filein_name, "rb" );

  if ( filein == NULL ) {
    printf ( "\n" );
    printf ( "BMP_READ - Fatal error!\n" );
    printf ( "  Could not open the input file.\n" );
    return ERROR;
  }
/*
  Read the header.
*/
  result = bmp_read_header ( filein, xsize, ysize, bsize, &psize );

  if ( result == ERROR ) {
    printf ( "\n" );
    printf ( "BMP_READ: Fatal error!\n" );
    printf ( "  BMP_READ_HEADER failed.\n" );
    return ERROR;
  }
/*
   Check for supported formats.
 */
  if ( (*bsize) !=8 && (*bsize) != 24)
  {
            printf ( "\n" );
     printf ( "BMP_READ: bit size = %d is not supported\n", *bsize);
     printf ( "  BMP_READ failed.\n" );
```

```
    return ERROR;
  }
  if (*bsize == 8)
        //force to read palette
        psize = 256;
/*
  Read the palette. This is dummy read.
*/
  result = bmp_read_palette ( filein, psize );

  if ( result == ERROR ) {
    printf ( "\n" );
    printf ( "BMP_READ: Fatal error!\n" );
    printf ( "  BMP_READ_PALETTE failed.\n" );
    return ERROR;
  }

/*
  Allocate storage.
*/
  numbytes = ( *xsize ) * ( *ysize ) * sizeof ( unsigned char );

  *rarray = ( unsigned char * ) malloc ( numbytes );
  if ( rarray == NULL ) {
    printf ( "\n" );
    printf ( "BMP_READ: Fatal error!\n" );
    printf ( "  Could not allocate data storage.\n" );
    return ERROR;
  }
  if (*bsize == 24)
  {
     *garray = ( unsigned char * ) malloc ( numbytes );
     if ( garray == NULL ) {
       printf ( "\n" );
       printf ( "BMP_READ: Fatal error!\n" );
       printf ( "  Could not allocate data storage.\n" );
       return ERROR;
     }

     *barray = ( unsigned char * ) malloc ( numbytes );
     if ( barray == NULL ) {
       printf ( "\n" );
       printf ( "BMP_READ: Fatal error!\n" );
       printf ( "  Could not allocate data storage.\n" );
       return ERROR;
  }
  }
/*
  Read the data.
*/

  result = bmp_read_data ( filein, *xsize, *ysize, *bsize, *rarray, *garray, *barray );

  if ( result == ERROR ) {
    printf ( "\n" );
    printf ( "BMP_READ: Fatal error!\n" );
```

```
    printf ( "  BMP_READ_DATA failed.\n" );
    return ERROR;
  }
/*
  Close the file.
*/
  fclose ( filein );

  return SUCCESS;
}

/******************************************************************************/

int bmp_read_data ( FILE *filein, int xsize, int ysize, int bsize,
                                        unsigned char *rarray, unsigned char *garray, unsigned char *barray
) {

/******************************************************************************/

/*
  Purpose:

    BMP_READ_DATA reads the image data of the BMP file.

  Discussion:

    On output, the RGB information in the file has been copied into the
    R, G and B arrays.

    Thanks to Peter Kionga-Kamau for pointing out an error in the
    previous implementation.

  Modified:

    31 May 2000

  Author:

    John Burkardt

  Parameters:

    Input, FILE *FILEIN, a pointer to the input file.

    Input, int XSIZE, YSIZE, the X and Y dimensions of the image.

          Input, int BSIZE, the number of bits per pixel. Supported 8 and 24 only

    Input, int *RARRAY, *GARRAY, *BARRAY, pointers to the red, green
    and blue color arrays.
*/
  int  i,j;
  unsigned char *indexb;
  unsigned char *indexg;
  unsigned char *indexr;
  int temp;
```

```
int  numbyte;

numbyte = 0;

for ( j = ysize-1; j >= 0; j-- ) {
          indexr = rarray + xsize*j*sizeof(unsigned char);
          indexg = garray + xsize*j*sizeof(unsigned char);
          indexb = barray + xsize*j*sizeof(unsigned char);

  for ( i = 0; i < xsize; i++ ) {
            if (bsize == 24)
            {

      temp = fgetc ( filein );
      if ( temp == EOF ) {
        printf ( "BMP_READ_DATA: Failed reading data byte %d.\n", numbyte );
        return ERROR;
      }
      *indexb= (unsigned char)temp;
      numbyte = numbyte + 1;
      indexb = indexb + 1;

      temp = fgetc ( filein );
      if ( temp == EOF ) {
        printf ( "BMP_READ_DATA: Failed reading data byte %d.\n", numbyte );
        return ERROR;
      }
      *indexg = (unsigned char)temp;
      numbyte = numbyte + 1;
      indexg = indexg + 1;

                        temp = fgetc ( filein );
      if ( temp == EOF ) {
        printf ( "BMP_READ_DATA: Failed reading data byte %d.\n", numbyte );
        return ERROR;
      }
      *indexr  = (unsigned char)temp;
      numbyte = numbyte + 1;
      indexr = indexr + 1;

            }
            else if (bsize == 8)
            {
                        temp = fgetc ( filein );
      if ( temp == EOF ) {
        printf ( "BMP_READ_DATA: Failed reading data byte %d.\n", numbyte );
        return ERROR;
      }
      *indexr = (unsigned char) temp;
      numbyte = numbyte + 1;
      indexr = indexr + 1;
            }
  }
  //re-align
  while(numbyte % 4)
  {
```

```
            fgetc(filein);
            numbyte++;
    }

  }

  return SUCCESS;
}
/******************************************************************************/

int bmp_read_header ( FILE *filein, int *xsize, int *ysize, int *bsize, int *psize ) {

/******************************************************************************/

/*
  Purpose:

    BMP_READ_HEADER reads the header information of a BMP file.

  Modified:

    05 October 1998

  Author:

    John Burkardt

  Parameters:

    Input, FILE *FILEIN, a pointer to the input file.

    Output, int *XSIZE, *YSIZE, the X and Y dimensions of the image.

    Output, int *bsize, the number of bit per pixel.

    Output, int *PSIZE, the number of colors in the palette.
*/
  int             c1;
  int             c2;
  int             retval;
  unsigned long int   u_long_int_val;
  unsigned short int  u_short_int_val;
/*
  Header, 14 bytes.
    2 bytes FileType;       Magic number: "BM",
    4 bytes FileSize;       Size of file in 32 byte integers,
    2 bytes Reserved1;      Always 0,
    2 bytes Reserved2;      Always 0,
    4 bytes BitmapOffset.   Starting position of image data, in bytes.
*/
  c1 = fgetc ( filein );
  if ( c1 == EOF ) {
    return ERROR;
  }
  c2 = fgetc ( filein );
  if ( c2 == EOF ) {
```

```
    return ERROR;
  }

  if ( c1 != 'B' || c2 != 'M' ) {
    return ERROR;
  }

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }

  retval = read_u_short_int ( &u_short_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }

  retval = read_u_short_int ( &u_short_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }
/*
  The bitmap header is 40 bytes long.
    4 bytes unsigned Size;           Size of this header, in bytes.
    4 bytes Width;                   Image width, in pixels.
    4 bytes Height;                  Image height, in pixels.  (Pos/Neg, origin at bottom, top)
    2 bytes Planes;                  Number of color planes (always 1).
    2 bytes BitsPerPixel;            1 to 24.  1, 4, 8 and 24 legal.  16 and 32 on Win95.
    4 bytes unsigned Compression;    0, uncompressed; 1, 8 bit RLE; 2, 4 bit RLE; 3, bitfields.
    4 bytes unsigned SizeOfBitmap;   Size of bitmap in bytes. (0 if uncompressed).
    4 bytes HorzResolution;          Pixels per meter. (Can be zero)
    4 bytes VertResolution;          Pixels per meter. (Can be zero)
    4 bytes unsigned ColorsUsed;     Number of colors in palette.  (Can be zero).
    4 bytes unsigned ColorsImportant. Minimum number of important colors. (Can be zero).
*/
  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }
  *xsize = ( int ) u_long_int_val;

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }
  *ysize = ( int ) u_long_int_val;
```

```
  retval = read_u_short_int ( &u_short_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }


  retval = read_u_short_int ( &u_short_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }
  *bsize = (int) u_short_int_val;

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }
  *psize = ( int ) u_long_int_val;

  retval = read_u_long_int ( &u_long_int_val, filein );
  if ( retval == ERROR ) {
    return ERROR;
  }


  return SUCCESS;
}
/*****************************************************************************/

int bmp_read_palette ( FILE *filein, int psize ) {

/*****************************************************************************/

/*
  Purpose:
```

```
  BMP_READ_PALETTE reads the palette information of a BMP file.
                  This really a dummy process. Just to make sure the
                                          data chunks are aligned correctly.

  Note:

  There are PSIZE colors listed.  For each color, the values of
  (B,G,R,A) are listed, where A is a quantity reserved for future use.

  Created:

  16 May 1999

  Author:

  John Burkardt

  Modified:

  Apr 05, 2003

  Author:

          Diem Vu
  Parameters:

  Input, FILE *FILEIN, a pointer to the input file.

  Input, int PSIZE, the number of colors in the palette.

  Remarks: Fixed the RGB read in order.
*/
  int  c;
  int  i;
  int  j;

  for ( i = 0; i < psize; i++ ) {
    for ( j = 0; j < 4; j++ ) {
      c = fgetc ( filein );
      if ( c == EOF ) {
        return ERROR;
      }
    }
  }

  return SUCCESS;
}


int bmp_write ( char *fileout_name, int xsize, int ysize, int bsize,
                                  unsigned char *rarray, unsigned char *garray, unsigned char *barray ) {

/******************************************************************************/

/*
  Purpose:
```

```
  BMP_WRITE writes the header and data for a BMP file.

  Created:

    02 October 1998

  Author:

    John Burkardt

  Modified:
     Apr 05, 2003

  Author:
          Diem Vu

  Parameters:

    Input, char *FILEOUT_NAME, the name of the output file.

    Input, int XSIZE, YSIZE, the X and Y dimensions of the image.

          Input, int bsize, number of bits per pixel (8 or 24)

    Input, unsigned *RARRAY, *GARRAY, *BARRAY, pointers to the red, green
    and blue color arrays.

   Remarks:

    If bsize = 8, only rarray will be used. The default palette will be created for the file (by
bmp_write_palette)
          to make it become grayscale format.
*/
  FILE *fileout;
  int   result;
/*
  Open the output file.
*/
  fileout = fopen ( fileout_name, "wb" );

  if ( fileout == NULL ) {
    printf ( "\n" );
    printf ( "BMP_WRITE - Fatal error!\n" );
    printf ( "  Could not open the output file.\n" );
    return ERROR;
  }
/*
  Write the header.
*/
  result = bmp_write_header ( fileout, xsize, ysize, bsize );

  if ( result == ERROR ) {
    printf ( "\n" );
    printf ( "BMP_WRITE: Fatal error!\n" );
    printf ( "  BMP_WRITE_HEADER failed.\n" );
```

```
    return ERROR;
  }
/*
  Write the palette if neccessary
*/
  if (bsize == 8)
  {
            result = bmp_write_palette ( fileout);

    if ( result == ERROR ) {
      printf ( "\n" );
      printf ( "BMP_WRITE: Fatal error!\n" );
      printf ( "  BMP_WRITE_PALETTE failed.\n" );
      return ERROR;
    }
  }
/*
  Write the data.
*/
  result = bmp_write_data ( fileout, xsize, ysize, bsize, rarray, garray, barray );

  if ( result == ERROR ) {
    printf ( "\n" );
    printf ( "BMP_WRITE: Fatal error!\n" );
    printf ( "  BMP_WRITE_DATA failed.\n" );
    return ERROR;
  }
/*
  Close the file.
*/
  fclose ( fileout );

  return SUCCESS;
}
/******************************************************************************/

int bmp_write_data ( FILE *fileout, int xsize, int ysize, int bsize,
                                          unsigned char *rarray, unsigned char *garray, unsigned char
*barray ) {

/******************************************************************************/

/*
  Purpose:

    BMP_WRITE_DATA writes the image data to the BMP file.

  Created:

    02 October 1998

  Author:

    John Burkardt

  Parameters:
```

```
    Input, FILE *FILEOUT, a pointer to the output file.

    Input, int XSIZE, YSIZE, the X and Y dimensions of the image.

            Input, int bsize, the number of bits per pixel (8 or 24)

    Input, unsigned char *RARRAY, *GARRAY, *BARRAY, pointers to the red, green
    and blue color arrays.
*/
  int  i;
  unsigned char *indexb;
  unsigned char *indexg;
  unsigned char *indexr;
  int  j;
  int numbyte;

  numbyte = 0;
  for (j = ysize-1; j>=0; j-- ) {
            /* Reverse the write order so the bitmap will be the
    the same as data array */

            indexr = rarray + xsize*j*sizeof(unsigned char);
            indexg = garray + xsize*j*sizeof(unsigned char);
            indexb = barray + xsize*j*sizeof(unsigned char);
    for ( i = 0; i < xsize; i++ ) {

              if (bsize == 24)
              {
       fputc ( *indexb, fileout );
                        fputc ( *indexg, fileout );
       fputc ( *indexr, fileout );
       indexb = indexb + 1;
       indexg = indexg + 1;
       indexr = indexr + 1;
                        numbyte+=3;
     }
     else
     {
            fputc ( *indexr, fileout );
            indexr = indexr + 1;
            numbyte++;
     }
    }
    //alligne
    while (numbyte %4)
    {
            fputc(0,fileout);
            numbyte++;
    }
  }

  return SUCCESS;
}
/*****************************************************************************/
int bmp_write_palette( FILE *fileout) {
```

```
/*
  Purpose:

    BMP_WRITE_PALETTE writes the dummy 256 palette information to a BMP file.

  Created:

    Apr 05, 2003

  Author:

    Diem Vu

  Parameters:

    Input, FILE *FILEOUT, a pointer to the output file.

    Input, int XSIZE, YSIZE, the X and Y dimensions of the image.
*/
          int      i;
          int      j;
    unsigned char c;

          for (i=0;i<256;i++)
          {
                    c = (unsigned char)i;
                    for(j=0;j<3;j++)
                              fputc ( c, fileout );
                    fputc(0,fileout);

          }
          return SUCCESS;
}
/*****************************************************************************/

int bmp_write_header ( FILE *fileout, int xsize, int ysize, int bsize ) {

/*****************************************************************************/

/*
  Purpose:

    BMP_WRITE_HEADER writes the header information to a BMP file.

  Created:

    02 October 1998

  Author:

    John Burkardt

  Parameters:

    Input, FILE *FILEOUT, a pointer to the output file.
```

```
   Input, int XSIZE, YSIZE, the X and Y dimensions of the image.
*/
  int               i;
  unsigned long int   u_long_int_val;
  unsigned short int  u_short_int_val;
  int xsize_aligned;

/*
  Header, 14 bytes.
    2 bytes FileType;       Magic number: "BM",
    4 bytes FileSize;       Size of file in bytes,
    2 bytes Reserved1;      Always 0,
    2 bytes Reserved2;      Always 0,
    4 bytes BitmapOffset.   Starting position of image data, in bytes.
*/
  fputc ( 'B', fileout );
  fputc ( 'M', fileout );


  if (bsize == 8)
  {
          xsize_aligned = xsize;
          while (xsize_aligned %4)
                  xsize_aligned++;
          u_long_int_val =  xsize_aligned * ysize + 54 + 256*4;
  }
  else
  {
          xsize_aligned = xsize;
          while (xsize_aligned %4)
                  xsize_aligned++;
          u_long_int_val = xsize_aligned * ysize + 54;
  }

  write_u_long_int ( u_long_int_val, fileout );

  u_short_int_val = 0;
  write_u_short_int ( u_short_int_val, fileout );

  u_short_int_val = 0;
  write_u_short_int ( u_short_int_val, fileout );


  if (bsize == 8)
  {
          u_long_int_val = 1078;
  }
  else
  {
          u_long_int_val = 54;
  }

  write_u_long_int ( u_long_int_val, fileout );
/*
  The bitmap header is 40 bytes long.
    4 bytes unsigned Size;          Size of this header, in bytes.
```

```
    4 bytes Width;                  Image width, in pixels.
    4 bytes Height;                 Image height, in pixels.  (Pos/Neg, origin at bottom, top)
    2 bytes Planes;                 Number of color planes (always 1).
    2 bytes BitsPerPixel;           1 to 24.  1, 4, 8 and 24 legal.  16 and 32 on Win95.
    4 bytes unsigned Compression;   0, uncompressed; 1, 8 bit RLE; 2, 4 bit RLE; 3, bitfields.
    4 bytes unsigned SizeOfBitmap;  Size of bitmap in bytes. (0 if uncompressed).
    4 bytes HorzResolution;         Pixels per meter. (Can be zero)
    4 bytes VertResolution;         Pixels per meter. (Can be zero)
    4 bytes unsigned ColorsUsed;    Number of colors in palette.  (Can be zero).
    4 bytes unsigned ColorsImportant. Minimum number of important colors. (Can be zero).
*/
  u_long_int_val = 40;
  write_u_long_int ( u_long_int_val, fileout );

  write_u_long_int ( xsize, fileout );

  write_u_long_int ( ysize, fileout );

  u_short_int_val = 1;
  write_u_short_int ( u_short_int_val, fileout );

  u_short_int_val = bsize;
  write_u_short_int ( u_short_int_val, fileout );

  u_long_int_val = 0;
  write_u_long_int ( u_long_int_val, fileout ); //compression

  u_long_int_val = (bsize/8)*xsize*ysize;
  write_u_long_int ( u_long_int_val, fileout );

  for ( i = 2; i < 4; i++ ) {
    u_long_int_val = 0;
    write_u_long_int ( u_long_int_val, fileout );
  }

  if (bsize == 8)
          u_long_int_val = 256;                 //Number of palette colors
  else
          u_long_int_val = 0;
  write_u_long_int ( u_long_int_val, fileout );

  u_long_int_val = 0;
  write_u_long_int ( u_long_int_val, fileout );

  return SUCCESS;
}
/******************************************************************************/

int bmp_write_test ( char *fileout_name ) {

/******************************************************************************/

/*
  Purpose:

    BMP_WRITE_TEST tests the BMP write routines.
```

```
  Modified:

    02 October 1998

  Author:

    John Burkardt

  Parameters:

    Input, char *FILEOUT_NAME, the name of the output file.
*/

  unsigned char *barray;
  unsigned char *garray;
  unsigned char *rarray;
  int  i;
  unsigned char *indexb;
  unsigned char *indexg;
  unsigned char *indexr;
  int  j;
  int  numbytes;

  int  result;
  int  xsize;
  int  ysize;
  int size = 120;
  xsize = size;
  ysize = size;
/*
  Allocate the memory.
*/
  rarray = NULL;
  garray = NULL;
  barray = NULL;
  numbytes = xsize * ysize * sizeof ( unsigned char  );

  rarray = ( unsigned char* ) malloc ( numbytes );

  if ( rarray == NULL ) {
    printf ( "\n" );
    printf ( "BMP_WRITE_TEST: Fatal error!\n" );
    printf ( "  Unable to allocate memory for data.\n" );
    return ERROR;
  }

  garray = ( unsigned char* ) malloc ( numbytes );

  if ( garray == NULL ) {
    printf ( "\n" );
    printf ( "BMP_WRITE_TEST: Fatal error!\n" );
    printf ( "  Unable to allocate memory for data.\n" );
    return ERROR;
  }
```

```
  barray = ( unsigned char * ) malloc ( numbytes );

  if ( barray == NULL ) {
    printf ( "\n" );
    printf ( "BMP_WRITE_TEST: Fatal error!\n" );
    printf ( "  Unable to allocate memory for data.\n" );
    return ERROR;
  }

  indexr = rarray;
  indexg = garray;
  indexb = barray;


  for (j = 0; j < ysize; j++ ) {
    for ( i = 0; i < xsize; i++ ) {
      if ( j < size/3 ) {
        *indexr = 255;
        *indexg = 0;
        *indexb = 0;
      }
      else if( j < size/3*2) {
        *indexr = 0;
        *indexg = 255;
        *indexb = 0;
      }
      else {
        *indexr = 0;
        *indexg = 0;
        *indexb = 255;
      }

      indexr = indexr + 1;
      indexg = indexg + 1;
      indexb = indexb + 1;
    }
  }
/*
  Write the data to a file.
*/
  result = bmp_write ( fileout_name, xsize, ysize, 24, rarray, garray, barray );
/*
  Free the memory.
*/
  if ( rarray != NULL ) {
    free ( rarray );
  }

  if ( garray != NULL ) {
    free ( garray );
  }

  if ( barray != NULL ) {
    free ( barray );
  }
```

```
  if ( result == ERROR ) {
    printf ( "\n" );
    printf ( "BMP_WRITE_TEST: Fatal error!\n" );
    printf ( "  BMP_WRITE failed.\n" );
    return ERROR;
  }

  return SUCCESS;
}
/******************************************************************************/

int read_u_long_int ( unsigned long int *u_long_int_val, FILE *filein ) {

/******************************************************************************/

/*
  Purpose:

    READ_U_LONG_INT reads an unsigned long int from FILEIN.

  Modified:

    20 May 2000

  Author:

    John Burkardt

  Parameters:

    Output, unsigned long int *U_LONG_INT_VAL, the value that was read.

    Input, FILE *FILEIN, a pointer to the input file.
*/
  int                retval;
  unsigned short int  u_short_int_val_hi;
  unsigned short int  u_short_int_val_lo;

  if ( byte_swap == TRUE ) {
    retval = read_u_short_int ( &u_short_int_val_lo, filein );
    if ( retval == ERROR ) {
      return ERROR;
    }
    retval = read_u_short_int ( &u_short_int_val_hi, filein );
    if ( retval == ERROR ) {
      return ERROR;
    }
  }
  else {
    retval = read_u_short_int ( &u_short_int_val_hi, filein );
    if ( retval == ERROR ) {
      return ERROR;
    }
    retval = read_u_short_int ( &u_short_int_val_lo, filein );
    if ( retval == ERROR ) {
      return ERROR;
```

```
    }
  }

/*
  Acknowledgement:

    A correction to the following line was supplied by
    Peter Kionga-Kamau, 20 May 2000.
*/

  *u_long_int_val = ( u_short_int_val_hi << 16 ) | u_short_int_val_lo;

  return SUCCESS;
}
/******************************************************************************/

int read_u_short_int ( unsigned short int *u_short_int_val, FILE *filein ) {

/******************************************************************************/

/*
  Purpose:

    READ_U_SHORT_INT reads an unsigned short int from FILEIN.

  Modified:

    16 May 1999

  Author:

    John Burkardt

  Parameters:

    Output, unsigned short int *U_SHORT_INT_VAL, the value that was read.

    Input, FILE *FILEIN, a pointer to the input file.
*/
  int chi;
  int clo;

  if ( byte_swap == TRUE ) {
    clo = fgetc ( filein );
    if ( clo == EOF ) {
      return ERROR;
    }
    chi = fgetc ( filein );
    if ( chi == EOF ) {
      return ERROR;
    }
  }
  else {
    chi = fgetc ( filein );
    if ( chi == EOF ) {
      return ERROR;
```

```
    }
    clo = fgetc ( filein );
    if ( clo == EOF ) {
      return ERROR;
    }
  }

  *u_short_int_val = ( chi << 8 ) | clo;

  return SUCCESS;
}

/******************************************************************************/

int write_u_long_int ( unsigned long int u_long_int_val, FILE *fileout ) {

/******************************************************************************/

/*
  Purpose:

    WRITE_U_LONG_INT writes an unsigned long int to FILEOUT.

  Modified:

    05 October 1998

  Author:

    John Burkardt

  Parameters:

    Input, unsigned long int *U_LONG_INT_VAL, the value to be written.

    Input, FILE *FILEOUT, a pointer to the output file.
*/
  unsigned short int  u_short_int_val_hi;
  unsigned short int  u_short_int_val_lo;

  u_short_int_val_hi = ( unsigned short ) ( u_long_int_val / 65536 );
  u_short_int_val_lo = ( unsigned short ) ( u_long_int_val % 65536 );

  if ( byte_swap == TRUE ) {
    write_u_short_int ( u_short_int_val_lo, fileout );
    write_u_short_int ( u_short_int_val_hi, fileout );
  }
  else {
    write_u_short_int ( u_short_int_val_hi, fileout );
    write_u_short_int ( u_short_int_val_lo, fileout );
  }

  return 4;
}

/******************************************************************************/
```

```
int write_u_short_int ( unsigned short int u_short_int_val, FILE *fileout ) {

/******************************************************************************/

/*
  Purpose:

    WRITE_U_SHORT_INT writes an unsigned short int to FILEOUT.

  Modified:

    05 October 1998

  Author:

    John Burkardt

  Parameters:

    Input, unsigned short int *U_SHORT_INT_VAL, the value to be written.

    Input, FILE *FILEOUT, a pointer to the output file.
*/
  unsigned char chi;
  unsigned char clo;

  chi = ( unsigned char ) ( u_short_int_val / 256 );
  clo = ( unsigned char ) ( u_short_int_val % 256 );

  if ( byte_swap == TRUE ) {
    fputc ( clo, fileout );
    fputc ( chi, fileout );
  }
  else {

    fputc ( chi, fileout );
    fputc ( clo, fileout );
  }

  return 2;
}



// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once


#include <iostream>
#include <tchar.h>
```

```
/*  bmp_io.h  16 May 1999  */

# define ERROR 1
# define SUCCESS 0

int bmp_read         ( char *filein_name, int *xsize, int *ysize, int* bsize,
                              unsigned char **rarray, unsigned char **garray, unsigned char **barray );
int bmp_read_data     ( FILE *filein, int xsize, int ysize, int bsize,
                                          unsigned char *rarray, unsigned char *garray, unsigned char *barray
);
int bmp_read_header   ( FILE *filein, int *xsize, int *ysize, int *bsize, int *psize );
int bmp_read_palette  ( FILE *filein, int psize );
int bmp_write_palette( FILE *fileout);
int bmp_read_test     ( char *filein_name );

int bmp_write        ( char *fileout_name, int xsize, int ysize, int bsize, unsigned char *rarray,
                  unsigned char *garray, unsigned char *barray );
int bmp_write_data    ( FILE *fileout, int xsize, int ysize, int bsize,
                                              unsigned char *rarray, unsigned char *garray, unsigned char
*barray);
int bmp_write_header  ( FILE *fileout, int xsize, int ysize, int bsize );
int bmp_write_test    ( char *fileout_name );

int read_u_long_int   ( unsigned long int *u_long_int_val, FILE *filein );
int read_u_short_int  ( unsigned short int *u_short_int_val, FILE *filein );

int write_u_long_int  ( unsigned long int u_long_int_val, FILE *fileout );
int write_u_short_int ( unsigned short int u_short_int_val, FILE *fileout );
```

END OF DOCUMENT