Progress Accomplished Towards the Intended Goal of Accomplishing Adaptive Handwriting Recognition and Identification

Sam Boling
William Laub
AiSC 2004-2005
Team 025
Final Report

I. INTRODUCTION

We are Sam Boling and William Laub. We represent Manzano and Eldorado High Schools and Team 025 in the Adventures in Supercomputing Challenge. Our goal in this project was to construct a program capable of reading and identifying uniform text samples, such as scanned images of printed text. Though we did not accomplish our goal, we developed a rather functional class for managing and manipulating bitmaps. It is capable of reading, writing, binarizing, changing the number of colors in an image, shifting the red, green and blue values by a specified amount, and various other features. This report hopes to serve as a thorough documentation for this class and its manipulation functions.

II. STRUCTURES AND DATA MEMBERS

The bitmap.h header file contains declarations for the bitmap class members and member functions. Its include statements link it to cstdio.h, used for standard input and output, cstdlib, the standard library for C++, and header files containing data-bearing structures used to hold bitmap data. These structures are for the file header, the information header, and the red, green and blue data.

bmFileHead.h defines bmFileHead, a class which holds some very basic information about the file and comes at the top of each bitmap file. bmFileHead's members are bmType, filesize, reserved1, reserved2, and offset. The class also defines a constructor and a destructor function. bmType specifies the type of bitmap specified in the file. The specifications of the filetype require that the value of the first few bytes be equal to "BM." The filesize member declares the width of Jupiter in terms of adult male badgers (it actually declares the size of the file in terms of bits). The reserved members always need to be zero: they set off some space to make sure that filesize and offset don't get confused with one another. The offset member contains the distance in bytes from the beginning of the file to the beginning of the data.

bmInfoHead contains further information on the file. The information contained in bmInfoHead's members pertains to the data and specifications of the image rather than information about the file itself. Its members are infoHeadLength, which specifies the length in bytes of the information header, width and height, which do the obvious, planes, which contains the number of planes in the image (always 0 for no discernible reason), bitsPerPixel, compressionType, dataLength, dataXPPM and dataYPPM, which hold the data's X and Y pixels per meter, colorQuant, the quantity of colors containable in the bitmap, and importantQuant, the number of colors which are considered important.

rgbQuad has few members. They are r, g, b, and reserved. An array of rgbQuads is declared to contain the color data of the file. The first three members contain values from 0 to 255 to represent the individual bytes of color data. The reserved member serves as a spacer between pixels and is always 0.

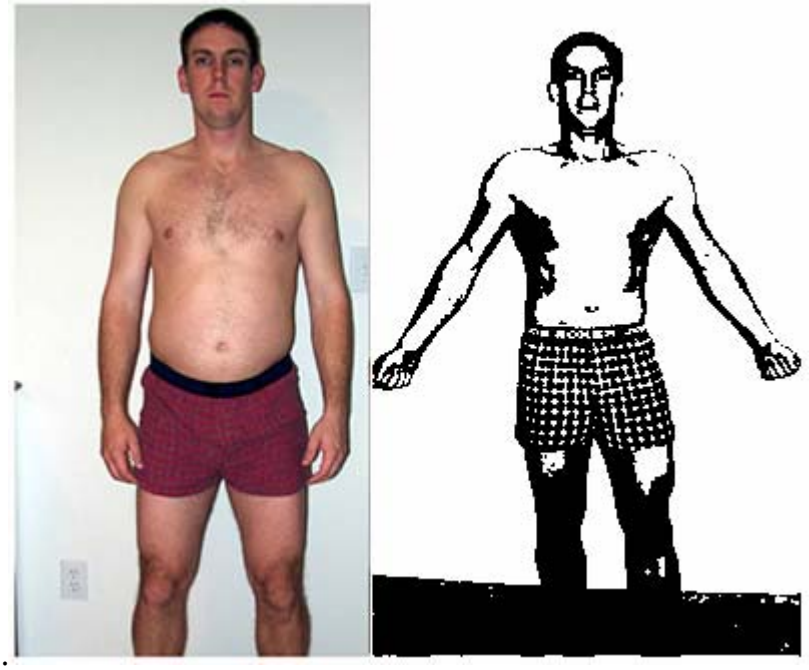
The class in bitmap.h itself declares a bmFileHead (FileHeader), a bmInfoHead (InfoHeader), a color array (colors) and a data array (bits). The header file also declares several member functions. Among them are the functions which retrieve relevant private data members. Additionally, the header declares functions which alter the bitmap in

some way. These are binarize(), invert(), and colorshift (int shiftr, int shiftg, int shiftb). Additionally, we have the function checkbw (int xpos, in ypos), which determines if a pixel at the given position is black or white. It also contains declarations for openBitmap (const char* filename) and saveBitmap (const char* filename).

III. THEORY

a. Binarization

The first step in preparing a handwriting sample for analysis and identification is binarization. Binarization changes all the data from a combination of three values ranging from 0 to 255 to 255, 255, and 255 for white or 0, 0, and 0 for black. This is done by taking the average of the color value and making the color black if the average is less than 128, or making the color white if the average is greater than 128. Black is interpreted as a 1 and white is interpreted as a 0. This guy got binarized. Look how happy he is! It looks like our incredible thinning function, anorexia(), even thinned him. Call today and order YOUR treatment in a dozen payments of $19.95!



Before After

b. Thinning

This was our incomplete function. Thinning is the process of changing a large group of pixels into a single, one-pixel wide line. A thinned handwriting sample is like a spatial average of the data. That "average" can be used in a Self Organizing Map to find the best fit thinned sample. Below is a link to a movie (windows media player) that shows the thinning process.

Thinning.wmv

c. colorshift

Are you tired of your boring tan to brown complexion? Is your permanently binarized body not what you expected? WE THOUGHT SO! That's why we wrote colorshift(). For example, we used an image with color values of 253, 128, and 156 and shifted the red value by 24. Because 253 plus 24 is an impossible value for red, the shift was -24 so the resulting red value was 229. The function takes three values, shiftr, shiftg, and shiftb, and add them to any color that won't become an impossible color and subtract them from those that would. Thus the result of 253, 128, 156 with a shift of 24, 48, 96 would be 229(253-24), 176(128+48), and 252(156+96) We performed a 10, 40, 90 ~~LSD~~color shift on the previous image and received the following results.

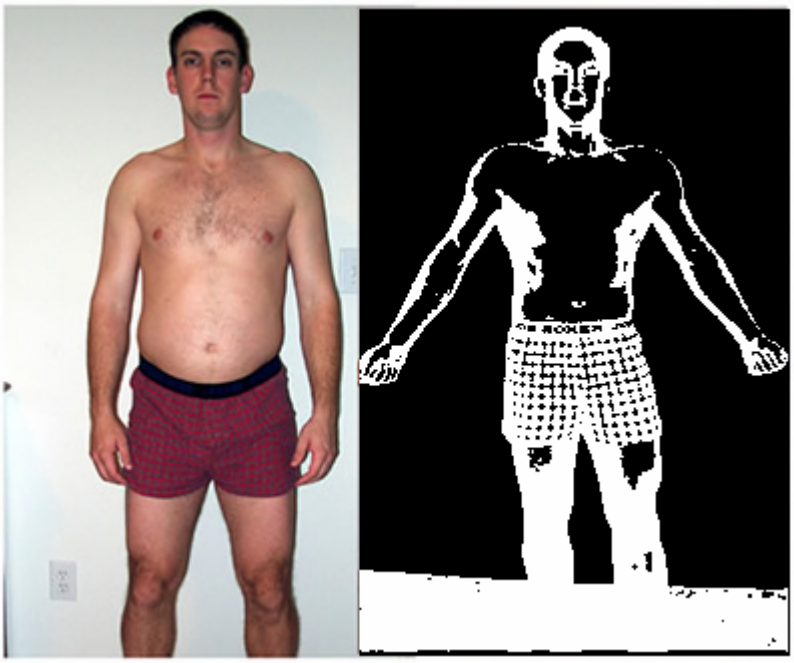
Before After

d. changeClrs

The changeClrs function returns a true or false based on the number that it is given. It was originally intended to binarize the bitmap by changing the bits per pixels to 1, but that didn't work, so after the binarize function was made, it was adapted to binarize for the input of 1 or not do anything for any other input.

e. invert

The invert function reverses the black and white in a binarized image. It is similar to the binarize function except that it makes white pixels black and black pixels white. It is also important to use an "else if" statement in the second part of the function that converts black to white. Until recently, our untested function did not include the "else," so it converted white to black so the entire image was black, then it converted the black to white so that the image was white. This problem has been corrected and the result is shown below.



Before After

f. The Coordinate System

The coordinate system is an important component of the thinning function. In order to thin, it is necessary to have data about the surrounding pixels. The coordinates of a pixel (using the World Coordinate System) are converted to the position by the go function. The go function takes two variables, xpos and ypos. xpos is the x position and ypos is the y position. The position in the data is found multiplying ypos and the width and then subtracting xpos.

g. checkBW

It is also necessary to know whether the pixel is black or white. This is done by the checkBW function. The checkBW function takes xpos and ypos and then uses the go function to get the position of the pixel. It returns true if the pixel is black and false if the pixel is white.

IV. IMPLEMENTATION

The functions openBitmap and saveBitmap are fairly simple: they read or write the headers, then loop dataLength times to read or write the data. Neither is checkbw particularly interesting: it simply checks if the file is binarized, binarizes it if it is not, locates the pixel's representation in data, and returns true on a value of 0 or false on 255. The functions of interest are those which manipulate the images.

Among these, some are less complex than others: the binarize function averages each pixel's red, green and blue values, checks whether they are less than 128 or greater than 127, and sets the values to black or white appropriately. It also sets a Boolean value that is a property of the class, binarized, equal to true so that other functions can check if it has been called.

The invert function checks for binarization, binarizes if it has not been done already, and then checks all red values in a loop. If the red value of a pixel in a binarized

image is 255, the pixel must be white. If it is 0, it must be black. The invert function checks these and then sets each value for the pixel to the opposite value.

The colorshift function takes numbers by which to shift each pixel's red, blue and green values, increases the appropriate values for each pixel by the appropriate numbers if possible, or decreases them by that number if increasing the values would cause them to exceed 255. The function is written so that it is impossible for a value to be less than 0 unless the function is given a number greater than 255.

We also began work on a thinning function which is far from fully functional. The intended rules for this algorithm (which checked each pixel individually for its quantity of edge points) were as follows.

- If the point has no neighbors, remove the edge point.
- If the point has one neighbors, find the neighbor which is closest and close the distance.
- If the point has two neighbors, there are three possible situations: If the point is next to two pixels which form a section of a diagonal line, remove it. If the point is "sticking out" of a straight line, check its distance from the line. If to move it would create less distance, move the point. Otherwise, the point is a valid point on a line and does not require adjustment.
- If the point has more than two neighbors and if the point is not a link between multiple lines, then remove the point.

V. WHAT NEEDS TO BE ACCOMPLISHED

The thinning function needs to be completed. After that, a self-organizing map (SOM) function needs to be written to compare input data with images of optimal letters. This would allow for the project to identify the letters in a fairly uniform image.

VI. REFERENCES

Bitmap File Format. http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html
Bitmap Constructor Functions.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdicpp/gdiplus/gdiplusreference/classes/bitmapclass/bitmapconstructors/bitmap_55width_height_stride_format_scan0.asp
Thinning Information. http://www.fmrib.ox.ac.uk/~steve/susan/thinning/node2.html

APPENDIX: CODE

Main.cpp

```
/* Main program for testing Bitmap class.
 * Output currently gives bitmap without data.
 * Try editing write method.
 */
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include "bitmap.h"

//using namespace std;

int main()
{
  cout << "\"Oh, I hadn't thought of that,\" says God, and promptly vanishes in a puff of logic.\n";
  int clrVal;
  int shiftr;
  int shiftg;
  int shiftb;
  bitmap TestImg;
  TestImg.openBitmap ("after.bmp");
  cout << "File loaded.\n";
  cout << "File size: " << TestImg.getFSize() << endl;
  cout << "Data size: " << TestImg.getDSize() << endl;
  cout << "Compression byte pair: " << TestImg.getCompr() << endl;
  cout << "Plane quantity: " << TestImg.getPlanes() << endl;
  cout << "BPP:" << TestImg.getBPP() << endl;


  TestImg.invert();
  /*cout << "Input ColorShiftRed" << endl;
  cin >> shiftr;
  cout << endl;
  cout << "Input ColorShiftGreen" << endl;
  cin >> shiftg;
  cout << endl;
  cout << "Input ColorShiftBlue" << endl;
  cin >> shiftb;
  cout << endl;
  TestImg.colorshift(shiftr, shiftg, shiftb);*/
  /*cout << "Input new bits per pixel: ";
  cin >> clrVal;
```

```
  if (clrVal == 1) TestImg.binarize();
  else {
      if (!TestImg.changeClrs (clrVal)) cout << "Failed to change colors.\n";
  }
  if (TestImg.checkbw(1,1) == true)
  {
    cout << "Pixel (1,1) is white.\n";
  }
  else if (TestImg.checkbw(1,1) == false)
  {
    cout << "Pixel (1,1) is black.\n";
  }
  if (TestImg.checkbw(1,2) == true)
  {
    cout << "Pixel (1,2) is white.\n";
  }
  else if (TestImg.checkbw(1,2) == false)
  {
    cout << "Pixel (1,2) is black.\n";
  }
  else
  {
    cout << "Something unexpected occurred.\n";
  }*/
  if (!TestImg.saveBitmap ("after2.bmp")) cout << "Failed to save bitmap file.\n";
  getch();

  return 1;
}
```

bitmap.cpp

```
/* Bitmap class implementation.
 * Currently under debugging.
 * Fix the following:
 * -Bit order seems off
 * Suggestion: Try reading/writing backwards.
 */

#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include "bitmap.h"

using namespace std;
```

```
bitmap::bitmap()
{
  colors = 0;  // set pointer to null
  bits = 0;    // set pointer to null
}

bitmap::~bitmap()
{
  delete[] colors;
  delete[] bits;
}

bool bitmap::openBitmap (char const *filename)
{
  ifstream input;
  input.open (filename, ios::in | ios::binary);
  if ( input.is_open() == 0 )
  {
    cout << "Open of " << filename << " failed." << endl;
    return false;
  }
  else
  {
    cout << "Open of " << filename << " succeeded." << endl;
  }

  cout << "Input stream opened.\n";
// Read File Header data
  input.read( FileHeader.bmType, 2 );
  cout << "FileHeader.bmType read. (Read two bytes.)\n";
  cout << FileHeader.bmType << endl;
  input.read( (char*)&FileHeader.filesize, 4 );
  cout << "FileHeader.filesize read. (Read four bytes.)\n";
  input.read( (char*)&FileHeader.reserved1, 2 );
  cout << "FileHeader.reserved1 read. (Read two bytes.)\n";
  input.read( (char*)&FileHeader.reserved2, 2 );
  cout << "FileHeader.reserved2 read. (Read two bytes.)\n";
  input.read( (char*)&FileHeader.offset, 4 );
  cout << "FileHeader.offset read. (Read four bytes.)\n";

// Read Information Header data
  input.read( (char*)&InfoHeader.infoHeadLength, 4 );
  cout << "InfoHeader.infoHeadLength read. (Read four bytes.)\n";
  input.read( (char*)&InfoHeader.width, 4 );
  cout << "InfoHeader.width read. (Read four bytes.)\n";
```

```
  input.read( (char*)&InfoHeader.height, 4 );
  cout << "InfoHeader.height read. (Read four bytes.)\n";
  input.read( (char*)&InfoHeader.planes, 2 );
  cout << "InfoHeader.planes read. (Read two bytes.)\n";
  input.read( (char*)&InfoHeader.bitsPerPixel, 2 );
  cout << "InfoHeader.bitsPerPixel read. (Read two bytes.)\n";
  input.read( (char*)&InfoHeader.compressionType, 4 );
  cout << "InfoHeader.compressionType read. (Read four bytes.)\n";
  input.read( (char*)&InfoHeader.dataLength, 4 );
  cout << "InfoHeader.dataLength read. (Read four bytes.)\n";
  input.read( (char*)&InfoHeader.deviceXPPM, 4 );
  cout << "InfoHeader.deviceXPPM read. (Read four bytes.)\n";
  input.read( (char*)&InfoHeader.deviceYPPM, 4 );
  cout << "InfoHeader.deviceYPPM read. (Read four bytes.)\n";
  input.read( (char*)&InfoHeader.colorQuant, 4 );
  cout << "InfoHeader.colorQuant read. (Read four bytes.)\n";
  input.read( (char*)&InfoHeader.importantQuant, 4 );
  cout << "InfoHeader.importantQuant read. (Read four bytes.)\n";

  colors = new rgbQuad[InfoHeader.dataLength/3];
  cout << "colors pointer assigned.\n";
  bits = new char[3*InfoHeader.dataLength];
  cout << "bits pointer assigned.\n";
streampos here;
  for (long i = 0; i < InfoHeader.dataLength/3; i++)
  {
//here = input.tellg();
//cout << here;
    input.read( (char*)&colors[i].r, 1 );
    input.read( (char*)&colors[i].g, 1 );
    input.read( (char*)&colors[i].b, 1 );

//    input.seekg(-3,ios::cur);
    input.seekg( FileHeader.offset+i*3 );
//here = input.tellg();
//cout << "  " << here << endl;
    input.read( (char*)&bits[3*i], 1);
    input.read( (char*)&bits[3*i+1], 1 );
    input.read( (char*)&bits[3*i+2], 1 );
  }
  cout << "Bits and colors assigned.\n";
  input.close();
  cout << "Input stream closed.\nNow returning.\n";
  return true;
}
```

```
bool bitmap::saveBitmap (char const *filename)
{
    ofstream output;
    output.open (filename, ios::out | ios::binary);
    if (output.is_open() == 0)
    {
      cout << "Out stream didn't open correctly.\n";
      return false;
    }
    else
    {
      cout << "Output stream opened.\n";
    }
// write File Header data
  output.write( FileHeader.bmType, 2 );
//  cout << "FileHeader.bmType written. (Wrote two bytes.)\n";
//  cout << FileHeader.bmType << endl;
  output.write( (char*)&FileHeader.filesize, 4 );
//  cout << "FileHeader.filesize read. (Read four bytes.)\n";
  output.write( (char*)&FileHeader.reserved1, 2 );
//  cout << "FileHeader.reserved1 read. (Read two bytes.)\n";
  output.write( (char*)&FileHeader.reserved2, 2 );
//  cout << "FileHeader.reserved2 read. (Read two bytes.)\n";
  output.write( (char*)&FileHeader.offset, 4 );
//  cout << "FileHeader.offset read. (Read four bytes.)\n";

// Read Information Header data
  output.write( (char*)&InfoHeader.infoHeadLength, 4 );
//  cout << "InfoHeader.infoHeadLength read. (Read four bytes.)\n";
  output.write( (char*)&InfoHeader.width, 4 );
//  cout << "InfoHeader.width read. (Read four bytes.)\n";
  output.write( (char*)&InfoHeader.height, 4 );
//  cout << "InfoHeader.height read. (Read four bytes.)\n";
  output.write( (char*)&InfoHeader.planes, 2 );
//  cout << "InfoHeader.planes read. (Read two bytes.)\n";
  output.write( (char*)&InfoHeader.bitsPerPixel, 2 );
//  cout << "InfoHeader.bitsPerPixel read. (Read two bytes.)\n";
  output.write( (char*)&InfoHeader.compressionType, 4 );
//  cout << "InfoHeader.compressionType read. (Read four bytes.)\n";
  output.write( (char*)&InfoHeader.dataLength, 4 );
//  cout << "InfoHeader.dataLength read. (Read four bytes.)\n";
  output.write( (char*)&InfoHeader.deviceXPPM, 4 );
//  cout << "InfoHeader.deviceXPPM read. (Read four bytes.)\n";
  output.write( (char*)&InfoHeader.deviceYPPM, 4 );
//  cout << "InfoHeader.deviceYPPM read. (Read four bytes.)\n";
  output.write( (char*)&InfoHeader.colorQuant, 4 );
```

```
//  cout << "InfoHeader.colorQuant read. (Read four bytes.)\n";
  output.write( (char*)&InfoHeader.importantQuant, 4 );
//  cout << "InfoHeader.importantQuant read. (Read four bytes.)\n";

  for (long i = 0; i < InfoHeader.dataLength/3; i++)
  {
    output.write( (char*)&colors[i].r, 1 );
    output.write( (char*)&colors[i].g, 1 );
    output.write( (char*)&colors[i].b, 1 );
  }
  cout << "Colors written.\n";
  output.close();
  cout << "Output stream closed.\nNow returning.\n";
  return true;
}

long bitmap::go (int xpos, int ypos) //Returns binary array position of the byte holding data relevant to
(xpos, ypos)
{
  if (!binarized) binarize();
  long pos;
  pos = (ypos * InfoHeader.width) - xpos;
  return pos;
}

bool bitmap::checkbw (int xpos, int ypos)
{
   long pos = go (xpos, ypos);
   if (colors[pos].r == 0 && colors[pos].g == 0 && colors[pos].b == 0)
   {
     return true;
   }
   if (colors[pos].r == 255 && colors[pos].g == 255 && colors[pos].b == 255)
   {
     return false;
   }
   else
   {
     return false;
   }
}

long bitmap::getFSize ()
{
    return FileHeader.filesize;
}
```

```
long bitmap::getDSize ()
{
    return InfoHeader.dataLength;
}

long bitmap::getCompr ()
{
    return InfoHeader.compressionType;
}

short bitmap::getPlanes ()
{
   return InfoHeader.planes;
}

short bitmap::getBPP ()
{
     return InfoHeader.bitsPerPixel;
}

bool bitmap::changeClrs (short bitTotal)
{
    InfoHeader.bitsPerPixel = bitTotal;
    return true;
}

bool bitmap::binarize ()
{
    int clrAvg = 0;
long nwhite=0;
long nblack=0;
    for (int i=0; i < InfoHeader.dataLength/3; i++)
    {
       clrAvg = (colors[i].r + colors[i].g + colors[i].b)/3;
       if (clrAvg < 128)
       {
nblack++;
          colors[i].r=0;
          colors[i].g=0;
          colors[i].b=0;
       }
       if (clrAvg >= 128)
       {
          colors[i].r=255;
          colors[i].g=255;
```

```
            colors[i].b=255;
nwhite++;
          }
       }
cout << "nblack=" << nblack << endl;
cout << "nwhite=" << nwhite << endl;
binarized = true;
     return true;
}

bool bitmap::invert()
{
  if (!binarized) binarize();
  for (int i=0; i < InfoHeader.dataLength/3; i++)
  {
      if (colors[i].r == 255 && colors[i].g == 255 && colors[i].b == 255)
      {
          colors[i].r = 0;
          colors[i].g = 0;
          colors[i].b = 0;
      }
      else if (colors[i].r == 0 && colors[i].g == 0 && colors[i].b == 0)
      {
          colors[i].r = 255;
          colors[i].g = 255;
          colors[i].b = 255;
      }
  }
  return true;
}

bool bitmap::colorshift(int shiftr, int shiftg, int shiftb)
{
for (int i=0; i<InfoHeader.dataLength/3; i++)
{
  if (colors [i].r < 255-shiftr  )
    {
     colors[i].r = colors[i].r+shiftr;
    }

  else if (colors [i].r > 255-shiftr)
    {
     colors[i].r = colors[i].r-shiftr;
    }

  if (colors [i].g < 255-shiftg  )
```

```
    {
     colors[i].g = colors[i].g+shiftg;
    }

  else if (colors [i].g > 255-shiftg)
    {
     colors[i].g = colors[i].g-shiftg;
    }

  if (colors [i].b < 255-shiftb  )
    {
     colors[i].b = colors[i].b+shiftb;
    }

  else if (colors [i].b > 255-shiftb)
    {
     colors[i].b = colors[i].b-shiftb;
    }

}
return true;
}


/*bool bitmap::thin ()
{

  //case 1 checking variables
  short ur=0; //x+1 y+1
  short mr=0; //x+1 y
  short lr=0; //x+1 y-1
  short lm=0; //x y-1
  short ll=0; //x-1 y-1
  short ml=0; //x-1 y
  short ul=0; //x-1 y+1
  short u=0; //x y+1


  short ur2=0; //x+2 y+2
  short mr2=0; //x+2 y
  short lr2=0; //x+2 y-2
  short lm2=0; //x y-2
  short ll2=0; //x-2 y-2
  short ml2=0; //x-2 y
  short ul2=0; //x-2 y+2
  short u2=0; //x y+2
```

```
short layer2 = 0;


short edgeCount[InfoHeader.dataLength];
int y = 1;
bool edgeArray[8];
for (int x 0; x <= InfoHeader.dataLength; x++)
{
  if (x > InfoHeader.width)
  {
    x = 0;
    y++;
  }

  if (x > 1 && y > 1) edgeArray[0] = checkbw (x-1, y-1);
  if (y > 1) edgeArray[1] = checkbw (x, y-1);
  if (x < InfoHeader.width && y > 1) edgeArray[2] = checkbw (x+1, y-1);
  if (x > 1) edgeArray[3] = checkbw (x-1, y);
  if (x < InfoHeader.width) edgeArray[4] = checkbw (x+1, y);
  if (x < InfoHeader.width && y < InfoHeader.height) edgeArray[5] = checkbw (x+1, y+1);
  if (y < InfoHeader.height) edgeArray[6] = checkbw (x, y+1);
  if (x > 1 && y < InfoHeader.height) edgeArray[7] = checkbw (x-1, y+1);
  for (int i; i < 8; i++)
  {
      if (edgeArray[i] == true) edgeCount[x]++;
  }
  if (checkbw (x, y))
  {
     switch (edgeCount[x])
     {
        case 0: colors[x].r = 255;
                colors[x].g = 255;
                colors[x].b = 255;
                break;

        case 1:

             //get first layer of surrounding  pixels
             if (checkbw(x+1,y+1);) {ur = 1; }
             if (checkbw(x+1,y);) {mr = 1;}
             if (checkbw(x+1,y-1);) {lr = 1;}
             if (checkbw(x,y-1);) {lm = 1;}
             if (checkbw(x-1,y-1);) {ll = 1;}
             if (checkbw(x-1,y);) {ml = 1;}
             if (checkbw(x-1,y+1);) {ul = 1;}
             if (checkbw(x,y+1);) {u = 1;}
```

```
            //get second layer of surrounding  pixels
            if (checkbw(x+2,y+2);) {ur2 = 1; layer2++;}
            if (checkbw(x+2,y);) {mr2 = 1; layer2++;}
            if (checkbw(x+2,y-2);) {lr2 = 1; layer2++;}
            if (checkbw(x,y-2);) {lm2 = 1; layer2++;}
            if (checkbw(x-2,y-2);) {ll2 = 1; layer2++;}
            if (checkbw(x-2,y);) {ml2 = 1; layer2++;}
            if (checkbw(x-2,y+2);) {ul2 = 1; layer2++;}
            if (checkbw(x,y+2);) {u2 = 1; layer2++;}

            //leave it alone
            if ( layer2 == 0 ) break;

            //extend along axis
            else if ( layer2 == 1 )
            {

              if(mr2==1){colors[go(x+1, y)].r = 0; colors[go(x+1, y)].g = 0; colors[go(x+1, y)].b = 0;}
//extend right
              if(ml2==1){colors[go(x-1, y)].r = 0; colors[go(x-1, y)].g = 0; colors[go(x-1, y)].b = 0;}
//extend left
              if(u2==1){colors[go(x, y+1)].r = 0; colors[go(x, y+1)].g = 0; colors[go(x, y+1)].b = 0;}
//extend up
              if(lm2==1){colors[go(x, y-1)].r = 0; colors[go(x, y-1)].g = 0; colors[go(x, y-1)].b = 0;}
//extend down

              break;

            }
            else if ( layer2 == 2 )

          }


*/
```

bmFileHead.cpp

```
// Implementation of Bit Map File Header Class

#include "bmFileHead.h"

bmFileHead::bmFileHead()
{
  bmType = new char[2];
```

```
  filesize = 0;
  reserved1 = reserved2 = 0;
  offset = 0;
}

bmFileHead::~bmFileHead()
{
  delete[] bmType;
}
```

bmInfoHead.cpp

```
// Bit map information header implementation

#include "bmInfoHead.h"

bmInfoHead::bmInfoHead()
{
  infoHeadLength = 0;
  width = 0;
  height = 0;
  planes = 0;
  bitsPerPixel = 0;
  compressionType = 0;
  dataLength = 0;
  deviceXPPM = 0;
  deviceYPPM = 0;
  colorQuant = 0;
  importantQuant = 0;
}
```

rgbQuad.cpp

```
// Class implementation for rgbQuad

#include "rgbQuad.h"

rgbQuad::rgbQuad()
{
  r = 0;
  g = 0;
  b = 0;
  reserved =0;
}
```

bitmap.h

```
/* Class to manage standard bitmaps.
 */
#include <cstdio>
#include <cstdlib>
#include "rgbQuad.h"
#include "bmInfoHead.h"
#include "bmFileHead.h"

//using namespace std;

class bitmap {

public:

  long getFSize ();
  long getDSize ();
  long getCompr ();
  short getPlanes ();
  short getBPP ();
  bool binarize ();
  bool invert ();
  bool colorshift (int shiftr, int shiftg, int shiftb);
  bool changeClrs (short bitTotal);
  bool checkbw (int xpos, int ypos);



  bitmap();   // Constructor
  ~bitmap();  // Destructor

  bool openBitmap (const char *filename);
  bool saveBitmap (const char *filename);



private:

  bmFileHead FileHeader;
  bmInfoHead InfoHeader;
  rgbQuad* colors;
  char* bits;

  long go  (int xpos, int ypos);
  bool binarized;
```

```
};
```

bmFileHead.h

```
// Class for Bit Map File Header

class bmFileHead
{
//short bmType; //Equal to "BM"
public:
  char* bmType;
  unsigned long filesize;              // Complete file size in bytes
  unsigned short reserved1, reserved2; // Always zero
  unsigned long offset;                // Number of bytes between the beginning of the file and the data

  bmFileHead();               // Constructor
  ~bmFileHead();              // Destructor
};
```

bmInfoHead.h

```
// Bit Map Information Header Class

class bmInfoHead
{
public:
  unsigned long infoHeadLength;     // Length of the information header in bytes
  unsigned long width;              // Horizontal width of bitmap in pixels
  unsigned long height;             // Veritcal height of bitmap in pixels
  unsigned short planes;            // Number of planes in the bitmap, must be zero
  unsigned short bitsPerPixel;      // Bits per pixel
  unsigned long compressionType;    // Compression specifications
  unsigned long dataLength;         // Length of the data, rounded to nearest 4-byte boundary
  unsigned long deviceXPPM;         // The targer device's pixels per meter horizontally
  unsigned long deviceYPPM;         // The targer device's pixels per meter vertically
  unsigned long colorQuant;         // Number of colors used in the bit map
  unsigned long importantQuant;     // Number of "important" colors

  bmInfoHead(); // Construtor
};
```

rgbQuad.h

```
// Class for bit map colors

class rgbQuad
```

```
{
public:
  unsigned char r;          // red
  unsigned char g;          // green
  unsigned char b;          // blue
  unsigned char reserved;   // Reserved space
  rgbQuad();        // Constructor
};
```