```
////////////////////////////////////////////////////////////////////////
//  CS75202 Computer Communication Networks
//
//  File:     vis.cpp
//  Author:   Kenneth W Schmidt
//  Abstract: This program listens for input in a listening thread for
//            users sending program to send either setup data or data
//            to be plotted on one of 9 scatter plots.  The main thread
//            plots points as a scatter plot as they are received on
//            each screen. The data in the array is normalized to
//            between 0 and 1 for consistent display. Normalization takes
//            place for each window after 10, 100 and 500 tuples are received for
//            a particular display window to keep the points visible within that
//            display window, and to refine the normalization twice after
//            a significant number of tuples are received.  New tuples
//            received after the array for each window is full replaces old
//            data to always display the newest MAXDATA points.  Note1:
//            global variables are used instead of message passing because
//            they are faster.  Note2: if this program is run on the same
//            host as the client, data transfer takes on the order of (up to) 1ms,
//            while if they are on different hosts, data transfer takes on
//            the order of (up to) 200ms (on a 100Mbps LAN), a tradeoff between
//            being able to send data quickly and being able to monitor remotely.
//  Revision History:   Date          Who      Description
//              ----------------------------------------------
//                      4/8/04              KWS            Initial Release
//                      4/20/04       KWS      Added "m" key to view scaling factors
////////////////////////////////////////////////////////////////////////

// for using WinMain() rather than main() in a Win32 application,
// reqired to turn off Win32 console window
#pragma comment (linker, "/ENTRY:WinMainCRTStartup")

// to turn off Win32 console window
#pragma comment (linker, "/subsystem:windows")

// must compile with this link module: Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#include "fstream.h"
#include "math.h"
#include "GL/glut.h"  // open GL
#include "stdio.h"
#include "winsock.h"  // for listenerThread socket connection
#include <stdarg.h>   // parse text and convert variables to text
#include <sstream>

#define MAX_PENDING 5 // max number of connections that can wait
#define MAXDATA 100000// max length of data arrays = max number of points to display on each window
#define SCALE1 10     // tuple no to perform the first scalling, should be 1% of MAXDATA
#define SCALE2 100    // tuple no to perform the second scalling, should be 10% of MAXDATA
#define SCALE3 50000  // tuple no to perform the second scalling, should be 50% of MAXDATA
#define MAXSTRING 1000// max string length for listener thread input
#define PORT 32767    // unassigned IANA port no for the listening thread to listen on
#define MAXNAME 50    // max length of the names of x and y axes
#define ERRORLENGTH 50// max length of error codes from GetLastError() from Windows

ifstream readFile;    // to read from a file

struct data
  {
    double x;         //point location in x plane
    double y;         //point location in y plane
  };

HANDLE handleListenerThread;  // handle for the listener thread so it can be closed by handle ID

char windowTitle1 [ MAXNAME ];// windowTitle(s) are the lables at the top of each data display window
char windowTitle2 [ MAXNAME ];
char windowTitle3 [ MAXNAME ];
```

```
char windowTitle4 [ MAXNAME ];
char windowTitle5 [ MAXNAME ];
char windowTitle6 [ MAXNAME ];
char windowTitle7 [ MAXNAME ];
char windowTitle8 [ MAXNAME ];
char windowTitle9 [ MAXNAME ];

char xTitle1 [ MAXNAME ];     // xTitle(s) and yTitle(s) are the lables for the x and y axes for each
  data display window
char yTitle1 [ MAXNAME ];
char xTitle2 [ MAXNAME ];
char yTitle2 [ MAXNAME ];
char xTitle3 [ MAXNAME ];
char yTitle3 [ MAXNAME ];
char xTitle4 [ MAXNAME ];
char yTitle4 [ MAXNAME ];
char xTitle5 [ MAXNAME ];
char yTitle5 [ MAXNAME ];
char xTitle6 [ MAXNAME ];
char yTitle6 [ MAXNAME ];
char xTitle7 [ MAXNAME ];
char yTitle7 [ MAXNAME ];
char xTitle8 [ MAXNAME ];
char yTitle8 [ MAXNAME ];
char xTitle9 [ MAXNAME ];
char yTitle9 [ MAXNAME ];

data dataArray1 [ MAXDATA ];  // array(s) for data for each screen received from the connected client
data dataArray2 [ MAXDATA ];
data dataArray3 [ MAXDATA ];
data dataArray4 [ MAXDATA ];
data dataArray5 [ MAXDATA ];
data dataArray6 [ MAXDATA ];
data dataArray7 [ MAXDATA ];
data dataArray8 [ MAXDATA ];
data dataArray9 [ MAXDATA ];

int n1 = 0;                   // length of each dataArray actually used
int n2 = 0;
int n3 = 0;
int n4 = 0;
int n5 = 0;
int n6 = 0;
int n7 = 0;
int n8 = 0;
int n9 = 0;

int np1 = 0;                  // pointer to the current position in each dataArray
int np2 = 0;                  // these are mod MAXDATA so the currently received data
int np3 = 0;                  // always replaces the oldest data in the dataArray
int np4 = 0;
int np5 = 0;
int np6 = 0;
int np7 = 0;
int np8 = 0;
int np9 = 0;

void display0 ( void );     // a blank window just to hold the windows open until data can be received
void display1 ( void );     // display functions draw the points of their respective screens that were
void display2 ( void );     // created by the init functions
void display3 ( void );
void display4 ( void );
void display5 ( void );
void display6 ( void );
void display7 ( void );
void display8 ( void );
void display9 ( void );

void help ( void );         // function to create a help MessageBox
```

```
void maxVal ( void );       // function to display max X and Y values

double x1Max = 1;           // xMax, yMax are for scaling so the
double y1Max = 1;           // scatter plot will fit in the visible
double x2Max = 1;           // window & to display the values with maxVal
double y2Max = 1;
double x3Max = 1;
double y3Max = 1;
double x4Max = 1;
double y4Max = 1;
double x5Max = 1;
double y5Max = 1;
double x6Max = 1;
double y6Max = 1;
double x7Max = 1;
double y7Max = 1;
double x8Max = 1;
double y8Max = 1;
double x9Max = 1;
double y9Max = 1;

                            // function to activate the keyboard to change screens and to exit the program
void keyboard ( unsigned char, int, int );

void init0 ();              // creates a small blank screen just to hold the windows open until data can be received
void init1 ();              // init functions initilaize the window size and location, set the window colors
void init2 ();              // set the window name, define the "visible world" and create two buffers to avoid
void init3 ();              // jitter when refreshing the display
void init4 ();
void init5 ();
void init6 ();
void init7 ();
void init8 ();
void init9 ();

int curWin = 0;             // which window are we currently using

                            // function to listen for input, receive and scale data.
DWORD WINAPI listenerThread( LPVOID );

char error [ERRORLENGTH];   // for error codes


////////////////////////////////////////////////////////////////////
//  main
//--------------------------------------------------------------------
//  Purpose:    main section of code
//            WINAPI WinMain used instead of main so that the console
//            window can be turned off since it is not needed for
//            interface
////////////////////////////////////////////////////////////////////

//int main(int argc, char** argv)
int WINAPI WinMain( HINSTANCE   hInstance,// Instance
          HINSTANCE hPrevInstance,      // Previous Instance
          LPSTR   lpCmdLine,            // Command Line Parameters
          int     nCmdShow)             // Window Show State
{
  MessageBox (NULL, "Please use the ESCAPE key to exit for proper cleanup", "On Exit",
  MB_SETFOREGROUND );


  // start the listener function in a new thread
  handleListenerThread =
    CreateThread
    (
```

```
      NULL,               // default security attributes used.
      0,                  // default stack size used.
      listenerThread,     // function name to run in new listener thread
      NULL,               // no argument passed to the listener thread function
      0,                  // no special flags are used.
      NULL                // no listener thread id requested.
    );

  if( handleListenerThread == NULL )
  {
    itoa (GetLastError(), error, 10);
    MessageBox (NULL, error, "Failed to create thread error", MB_SETFOREGROUND );
    return 0;
  }

  init0 ();
  help ();            //display the help box
  glutMainLoop();     //waiting for an event - click of mouse, etc.

  return 0;
}
////////////////////////////////////////////////////////////////////
//  init0
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer blank window to hold the draw
//            windows open until data can be received for data windows 1-9
////////////////////////////////////////////////////////////////////

void init0 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                  //GLUT_DOUBLE is 2 buffer, prevents jittery display
                                  //GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    //position of upper left corner of window
  glutInitWindowSize(300,300);      //size in pixels
  glutCreateWindow ( "This Window Intentionally Blank" );//create a blank window

  glClearColor(0.0, 0.0, 0.0, 0.0); //open a blank GL command window,
                                  //choose the background color
                                  //R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  //project the image to the front plane
  glLoadIdentity();             //loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display0 ); //draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );//activate the keyboard function
}


////////////////////////////////////////////////////////////////////
//  init1
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 1st window
////////////////////////////////////////////////////////////////////

void init1 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                  // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                  // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle1 );// create window with readFileName title of window

  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
```

```
                                    // choose the background color
                                    // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display1 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
  glutIdleFunc ( display1 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  init2
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 2nd window
////////////////////////////////////////////////////////////////////

void init2 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                    // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                    // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle2 );// create window with readFileName title of window

  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
                                    // choose the background color
                                    // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display2 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
  glutIdleFunc ( display2 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  init3
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 3rd window
////////////////////////////////////////////////////////////////////

void init3 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                    // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                    // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle3 );// create window with readFileName title of window

  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
                                    // choose the background color
                                    // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
```

```
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display3 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
  glutIdleFunc ( display3 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  init4
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 4th window
////////////////////////////////////////////////////////////////////

void init4 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                  // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                  // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle4 );// create window with readFileName title of window

  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
                                  // choose the background color
                                  // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display4 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
  glutIdleFunc ( display4 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  init5
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 5th window
////////////////////////////////////////////////////////////////////

void init5 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                  // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                  // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle5 );// create window with readFileName title of window

  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
                                  // choose the background color
                                  // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display5 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
```

```
  glutIdleFunc ( display5 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  init6
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 6th window
////////////////////////////////////////////////////////////////////

void init6 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                  // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                  // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle6 );// create window with readFileName title of window

  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
                                  // choose the background color
                                  // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display6 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
  glutIdleFunc ( display6 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  init7
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 7th window
////////////////////////////////////////////////////////////////////

void init7 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                  // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                  // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle7 );// create window with readFileName title of window


  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
                                  // choose the background color
                                  // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display7 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
  glutIdleFunc ( display7 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  init8
```

```
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 8th window
////////////////////////////////////////////////////////////////////

void init8 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                  // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                  // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle8 );// create window with readFileName title of window

  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
                                  // choose the background color
                                  // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display8 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
  glutIdleFunc ( display8 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  init9
//-----------------------------------------------------------------
//  Purpose:  creates a single buffer drawing window for the input
//            points for the 9th window
////////////////////////////////////////////////////////////////////

void init9 ()
{
  glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
                                  // GLUT_DOUBLE is 2 buffer, prevents jittery display
                                  // GLUT_RGB means we are using colors
  glutInitWindowPosition(25,25);    // position of upper left corner of window
  glutInitWindowSize(625,625);      // size in pixels
  glutCreateWindow ( windowTitle9 );// create window with readFileName title of window

  glClearColor(0.0, 0.0, 0.0, 0.0); // open a blank GL command window,
                                  // choose the background color
                                  // R, G, Blu, Transparancy alpha coefficient

  glMatrixMode(GL_PROJECTION);  // project the image to the front plane
  glLoadIdentity();             // loads a 4x4 identity matrix for normal initial projection
  glOrtho(-0.1, 1.5, -0.1, 1.5, -1.0, 1.0); //visible world, projection mode
        //  x    x    y     y     z    z, lower and upper limits of x,y,z
        //glOrtho says take the image from the 0 in the Z plane and project it
        //to the front plane (which would be the max pos limit of Z

  glutDisplayFunc ( display9 ); // draw the initial image, display is another function
  glutKeyboardFunc ( keyboard );// activate the keyboard function
  glutIdleFunc ( display9 );    // keep refreshing the active window since new points will be arriving
}

////////////////////////////////////////////////////////////////////
//  display0
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray1 on screen 1
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////
```

```
void display0 ( )
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object

  glLineWidth (10.0);           // make the big X fat

  glBegin(GL_LINES);            // draw a big X
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 1.5, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
  glEnd();

  glutSwapBuffers ();    //swap to buffer that is finished with calculating its dispaly
}


////////////////////////////////////////////////////////////////////
//  display1
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray1 on screen 1
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display1 ( )
{
  int i = 0;                    //loop counter
  glClear( GL_COLOR_BUFFER_BIT);//buffer for background, colors the background
                                //using color black
  glColor3f(1.0, 1.0, 1.0);     //R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           //size of points in pixels
  glBegin(GL_POINTS);           //draw a point for every glVertex3f until glEnd()

  for ( i = 0; i < n1; i++ )    //draw points on screen from array
  {
    glVertex3f ( dataArray1 [ i ].x, dataArray1 [ i ].y, 0.0 );
  }

  glEnd();                      //end of glBegin()

  glBegin(GL_LINES);            //draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      //swap to buffer that is finished with calculating its dispaly
}

////////////////////////////////////////////////////////////////////
//  display2
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray2 on screen 2
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display2 ( )
```

```
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           // size of points in pixels
  glBegin(GL_POINTS);           // draw a point for every glVertex3f until glEnd()

  for ( i = 0; i < n2; i++ )    // draw points on screen from array
  {
    glVertex3f ( dataArray2 [ i ].x, dataArray2 [ i ].y, 0.0 );
  }

  glEnd();                      // end of glBegin()

  glBegin(GL_LINES);            // draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      // swap to buffer that is finished with calculating its dispaly
}

////////////////////////////////////////////////////////////////////
//  display3
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray3 on screen 3
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display3 ( )
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           // size of points in pixels
  glBegin(GL_POINTS);           // draw a point for every glVertex3f until glEnd()

  for ( i = 0; i < n3; i++ )    // draw points on screen from array
  {
    glVertex3f ( dataArray3 [ i ].x, dataArray3 [ i ].y, 0.0 );
  }

  glEnd();                      // end of glBegin()

  glBegin(GL_LINES);            // draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      // swap to buffer that is finished with calculating its dispaly
}

////////////////////////////////////////////////////////////////////
//  display4
```

```
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray4 on screen 4
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display4 ( )
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           // size of points in pixels
  glBegin(GL_POINTS);           // draw a point for every glVertex3f until glEnd()

  for ( i = 0; i < n4; i++ )    // draw points on screen from array
  {
    glVertex3f ( dataArray4 [ i ].x, dataArray4 [ i ].y, 0.0 );
  }

  glEnd();                      // end of glBegin()

  glBegin(GL_LINES);            // draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      // swap to buffer that is finished with calculating its dispaly
}

////////////////////////////////////////////////////////////////////
//  display5
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray5 on screen 5
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display5 ( )
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           // size of points in pixels
  glBegin(GL_POINTS);           // draw a point for every glVertex3f until glEnd()

  for ( i = 0; i < n5; i++ )    // draw points on screen from array
  {
    glVertex3f ( dataArray5 [ i ].x, dataArray5 [ i ].y, 0.0 );
  }

  glEnd();                      // end of glBegin()

  glBegin(GL_LINES);            // draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
```

```
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      // swap to buffer that is finished with calculating its dispaly
}

////////////////////////////////////////////////////////////////////
//  display6
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray6 on screen 6
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display6 ( )
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           // size of points in pixels
  glBegin(GL_POINTS);           // draw a point for every glVertex3f until glEnd()

  for ( i = 0; i < n6; i++ )    // draw points on screen from array
  {
    glVertex3f ( dataArray6 [ i ].x, dataArray6 [ i ].y, 0.0 );
  }

  glEnd();                      // end of glBegin()

  glBegin(GL_LINES);            // draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      // swap to buffer that is finished with calculating its dispaly
}

////////////////////////////////////////////////////////////////////
//  display7
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray7 on screen 7
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display7 ( )
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           // size of points in pixels
  glBegin(GL_POINTS);           // draw a point for every glVertex3f until glEnd()

  for ( i = 0; i < n7; i++ )    // draw points on screen from array
  {
    glVertex3f ( dataArray7 [ i ].x, dataArray7 [ i ].y, 0.0 );
  }

  glEnd();                      // end of glBegin()
```

```
  glBegin(GL_LINES);            // draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      // swap to buffer that is finished with calculating its dispaly
}

////////////////////////////////////////////////////////////////////
//  display8
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray8 on screen 8
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display8 ( )
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           // size of points in pixels
  glBegin(GL_POINTS);           // draw a point for every glVertex3f until glEnd()

  for ( i = 0; i < n8; i++ )    // draw points on screen from array
  {
    glVertex3f ( dataArray8 [ i ].x, dataArray8 [ i ].y, 0.0 );
  }

  glEnd();                      // end of glBegin()

  glBegin(GL_LINES);            // draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      // swap to buffer that is finished with calculating its dispaly
}

////////////////////////////////////////////////////////////////////
//  display9
//-----------------------------------------------------------------
//  Purpose:  sets the color to white, point size to 2 pixels, plots
//            the points from dataArray9 on screen 9
//  Input:    nothing
//  Output:   the points have been plotted on the screen
////////////////////////////////////////////////////////////////////

void display9 ( )
{
  int i = 0;                    // loop counter
  glClear( GL_COLOR_BUFFER_BIT);// buffer for background, colors the background
                                // using color black
  glColor3f(1.0, 1.0, 1.0);     // R, G, Blu, 1,1,1 is white, color of object
  glPointSize( 2.0 );           // size of points in pixels
  glBegin(GL_POINTS);           // draw a point for every glVertex3f until glEnd()
```

```
  for ( i = 0; i < n9; i++ )    // draw points on screen from array
  {
    glVertex3f ( dataArray9 [ i ].x, dataArray9 [ i ].y, 0.0 );
  }

  glEnd();                      // end of glBegin()

  glBegin(GL_LINES);            // draw 2 axes
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.5, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.5, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -0.025, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(-0.025, 1.0, 0.0);
  glEnd();

  glutSwapBuffers ();      // swap to buffer that is finished with calculating its dispaly
}



////////////////////////////////////////////////////////////////////
//  help
//-----------------------------------------------------------------
//  Purpose:  displays a help message box
//  Input:    nothing
//  Output:   pops up a help message box
////////////////////////////////////////////////////////////////////

void help ()
{
  MessageBox (NULL, "'h' key:                  displays this help screen\n'escape' key           to ↙
  quit the program \n'm' key                   to see max values\n'1' key                          ↙
  displays screen 1\n'2' key                    displays screen 2\n'3' key                         ↙
  displays screen 3\n'4' key                    displays screen 4\n'5' key                         ↙
  displays screen 5\n'6' key                    displays screen 6\n'7' key                         ↙
  displays screen 7\n'8' key                    displays screen 8\n'9' key                         ↙
  displays screen 9", "Help", MB_SETFOREGROUND );
}

////////////////////////////////////////////////////////////////////
//  maxVal
//-----------------------------------------------------------------
//  Purpose:  displays a max values message box
//  Input:    nothing
//  Output:   pops up a max values message box
////////////////////////////////////////////////////////////////////
void maxVal ()
{
  std::stringstream maxMsg;       // message showing max X and Y values
  switch ( curWin )
  {
    case 1:
      maxMsg << "Max X = " << x1Max << "\nMax Y = " << y1Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;

    case 2:
      maxMsg << "Max X = " << x2Max << "\nMax Y = " << y2Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;

    case 3:
      maxMsg << "Max X = " << x3Max << "\nMax Y = " << y3Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;
```

```
    case 4:
      maxMsg << "Max X = " << x4Max << "\nMax Y = " << y4Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;

    case 5:
      maxMsg << "Max X = " << x5Max << "\nMax Y = " << y5Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;

    case 6:
      maxMsg << "Max X = " << x6Max << "\nMax Y = " << y6Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;

    case 7:
      maxMsg << "Max X = " << x7Max << "\nMax Y = " << y7Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;

    case 8:
      maxMsg << "Max X = " << x8Max << "\nMax Y = " << y8Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;

    case 9:
      maxMsg << "Max X = " << x9Max << "\nMax Y = " << y9Max;
      MessageBox (NULL, maxMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );
      break;
  }
}

////////////////////////////////////////////////////////////////////
//  GLUT callback function keyboard
//--------------------------------------------------------------------
//  Purpose:  activates keyboard, exits program for escape key, calls
//            the help message box if 'h' is pressed, changes to 2-D
//            display if the number 2 is pressed, changes to 3-D display
//            if the number 3 is pressed.
////////////////////////////////////////////////////////////////////

void keyboard ( unsigned char key, int x, int y )
{
  switch ( key )
  {
    case 27:  // escape key
      CloseHandle(handleListenerThread); // kill the listener thread before exiting
      exit ( 0 );
      break;

    case 'h':
      help ();
      break;

    case 'm':
      maxVal ();
      break;

    case '1':
      init1 ();
      curWin = 1;
      break;

    case '2':
      init2 ();
      curWin = 2;
      break;

    case '3':
      init3 ();
```

```
      curWin = 3;
      break;

    case '4':
      init4 ();
      curWin = 4;
      break;

    case '5':
      init5 ();
      curWin = 5;
      break;

    case '6':
      init6 ();
      curWin = 6;
      break;

    case '7':
      init7 ();
      curWin = 7;
      break;

    case '8':
      init8 ();
      curWin = 8;
      break;

    case '9':
      init9 ();
      curWin = 9;
      break;

    default: break;
  }
}

//////////////////////////////////////////////////////////////////////
//  listenerThread
//------------------------------------------------------------------
//  Purpose:  listens for connection from a client, receives input tuples
//            if they are window lables, adds them to the appropriate variables
//            if they are values, adds values to the appropriate array
//            sets the limit pointers for the arrays, scales values
//            in arrays for the scatter plots to fit in the visible windows
//  Input:    connection from client, input tuples from client
//  Output:   tuples of data in appropriate global array variables, scaled
//////////////////////////////////////////////////////////////////////
DWORD WINAPI listenerThread( LPVOID )
{
  int acceptLength;                  // Length of received address
  int clientSocRetVal = 0;           // Return value of recv function
  int dataType;                      // type of data received ( 0 = setup, 1 = data )
  int screenNo;                      // no of the screen that data is being sent to
  int i;                             // loop counter
  char buffer[MAXSTRING] = "";       // Buffer to send socket error message to client
  char clientData[MAXSTRING] = "";   // data from client
  char xVal[MAXSTRING] = "";         // value of x data or x setup data received
  char yVal[MAXSTRING] = "";         // value of y data or y setup data received

  double x1 = 0;                     // values of x, y received for plotting
  double y1 = 0;
  double x2 = 0;
  double y2 = 0;
  double x3 = 0;
  double y3 = 0;
  double x4 = 0;
  double y4 = 0;
  double x5 = 0;
  double y5 = 0;
```

```
  double x6 = 0;
  double y6 = 0;
  double x7 = 0;
  double y7 = 0;
  double x8 = 0;
  double y8 = 0;
  double x9 = 0;
  double y9 = 0;

  double oldx1Max = 1;              // keep the old value to correct new scaling value
  double oldy1Max = 1;
  double oldx2Max = 1;
  double oldy2Max = 1;
  double oldx3Max = 1;
  double oldy3Max = 1;
  double oldx4Max = 1;
  double oldy4Max = 1;
  double oldx5Max = 1;
  double oldy5Max = 1;
  double oldx6Max = 1;
  double oldy6Max = 1;
  double oldx7Max = 1;
  double oldy7Max = 1;
  double oldx8Max = 1;
  double oldy8Max = 1;
  double oldx9Max = 1;
  double oldy9Max = 1;


  // Initilaize sockets to invalid so we can check to
  // see if a valid socket connection has been made
  SOCKET winSock = INVALID_SOCKET, clientSock = INVALID_SOCKET;
  SOCKADDR_IN localSin, acceptSin;  // Address of local and client socket
  WSADATA WSAData;                  // A struct of winSock data for WS2-32.dll

  // Initialize winSock.
  if ( WSAStartup (MAKEWORD(1,1), &WSAData) != 0 )
  {
    itoa (WSAGetLastError (), error, 10);
    MessageBox (NULL, error, "Error initializing winSock", MB_SETFOREGROUND );
    return FALSE;
  }

  // Create a TCP/IP socket
  if ( ( winSock = socket ( AF_INET, SOCK_STREAM, 0 ) ) == INVALID_SOCKET )
  {
    itoa (WSAGetLastError (), error, 10);
    MessageBox (NULL, error, "Error, allocating the socket", MB_SETFOREGROUND );
    return FALSE;
  }

  // Fill out the local socket's address information.
  localSin.sin_family = AF_INET;
  localSin.sin_port = htons ( PORT );
  localSin.sin_addr.s_addr = htonl ( INADDR_ANY );

  // Bind the local address with winSock.
  if ( bind ( winSock, ( struct sockaddr * ) &localSin, sizeof ( localSin ) ) == SOCKET_ERROR )
  {
    itoa (WSAGetLastError (), error, 10);
    MessageBox (NULL, error, "Error binding socket", MB_SETFOREGROUND );
    closesocket (winSock);
    return FALSE;
  }

  // Create a socket to listen for client connections.
  if ( listen (winSock, MAX_PENDING) == SOCKET_ERROR )
  {
    itoa (WSAGetLastError (), error, 10);
    MessageBox (NULL, error, "Error listening to client", MB_SETFOREGROUND );
```

```
  }
  acceptLength = sizeof (acceptSin);

  // Accept an incoming client connection
  clientSock = accept ( winSock, ( struct sockaddr * ) &acceptSin, ( int * ) &acceptLength );

  // Stop listening for connections from clients. Only one connection at a time
  closesocket ( winSock );

  // Check for invalid socket errors
  if ( clientSock == INVALID_SOCKET )
  {
    itoa (WSAGetLastError (), error, 10);
    MessageBox (NULL, error, "Error accepting a connection from the client", MB_SETFOREGROUND );
  }

  // get data from client
  while (true)
  {
    // Receive data type from the client ( 0 = setup, 1 = data )
    clientSocRetVal = recv ( clientSock, clientData, MAXSTRING, 0 );

//std::stringstream aMsg;       // message showing max X and Y values
//aMsg << "clientData = " << clientData ;
//MessageBox (NULL, aMsg.str().c_str(), "Maximum Values", MB_SETFOREGROUND );


    //MessageBox (NULL, LPCTSTR(clientData), "Maximum Values", MB_SETFOREGROUND );
    // Is there an error connecting to the client socket
    if ( clientSocRetVal == SOCKET_ERROR )
    {
      itoa (WSAGetLastError (), error, 10);
      MessageBox (NULL, error, "Error, connecting to client", MB_SETFOREGROUND );
    }

    // Data is received from the client OK, display the data
    else
    {
      dataType = atoi ( clientData );
    }

    // send an ack "x" to the client so they send the next data
    if (send ( clientSock, "x", 2, 0) == SOCKET_ERROR )
    {
      itoa (WSAGetLastError (), error, 10);
      MessageBox (NULL, error, "Error, sending data to the client failed", MB_SETFOREGROUND );
    }

    // Receive data screen no from the client ( data goes to screen 1 through 9 )
    clientSocRetVal = recv ( clientSock, clientData, MAXSTRING, 0 );
    //MessageBox (NULL, LPCTSTR(clientData), "Maximum Values", MB_SETFOREGROUND );
    // Is there an error connecting to the client socket
    if ( clientSocRetVal == SOCKET_ERROR )
    {
      itoa (WSAGetLastError (), error, 10);
      MessageBox (NULL, error, "Error, connecting to client", MB_SETFOREGROUND );
    }

    // Data is received from the client OK, display the data
    else
    {
      screenNo = atoi ( clientData );
    }

    // send an ack "x" to the client so they send the next data
    if (send ( clientSock, "x", 2, 0) == SOCKET_ERROR )
    {
      itoa (WSAGetLastError (), error, 10);
      MessageBox (NULL, error, "Error, sending data to the client failed", MB_SETFOREGROUND );
    }
```

```
    // Receive x value ( data ) or x name ( setup ) from the client.
    clientSocRetVal = recv ( clientSock, clientData, MAXSTRING, 0 );
    //MessageBox (NULL, LPCTSTR(clientData), "Maximum Values", MB_SETFOREGROUND );
    // Is there an error connecting to the client socket
    if ( clientSocRetVal == SOCKET_ERROR )
    {
      itoa (WSAGetLastError (), error, 10);
      MessageBox (NULL, error, "Error, connecting to client", MB_SETFOREGROUND );
    }

    // Data is received from the client OK, display the data
    else
    {
      strcpy ( xVal, clientData );
    }

    // send an ack "x" to the client so they send the next data
    if (send ( clientSock, "x", 2, 0) == SOCKET_ERROR )
    {
      itoa (WSAGetLastError (), error, 10);
      MessageBox (NULL, error, "Error, sending data to the client failed", MB_SETFOREGROUND );
    }

    // Receive y value ( data ) or y name ( setup ) from the client.
    clientSocRetVal = recv ( clientSock, clientData, MAXSTRING, 0 );
    //MessageBox (NULL, LPCTSTR(clientData), "Maximum Values", MB_SETFOREGROUND );
    // Is there an error connecting to the client socket
    if ( clientSocRetVal == SOCKET_ERROR )
    {
      itoa (WSAGetLastError (), error, 10);
      MessageBox (NULL, error, "Error, connecting to client", MB_SETFOREGROUND );
    }

    // Data is received from the client OK, display the data
    else
    {
      strcpy ( yVal, clientData );
    }

    // dataType = 0: setup
    if ( dataType == 0 )
    {
      switch ( screenNo )
      {
        case 1:
          strcpy ( xTitle1, xVal );
          strcpy ( yTitle1, yVal );
          strcpy ( windowTitle1, xVal );
          strcat ( windowTitle1, " vs " );
          strcat ( windowTitle1, yVal );
          break;

        case 2:
          strcpy ( xTitle2, xVal );
          strcpy ( yTitle2, yVal );
          strcpy ( windowTitle2, xVal );
          strcat ( windowTitle2, " vs " );
          strcat ( windowTitle2, yVal );
          break;

        case 3:
          strcpy ( xTitle3, xVal );
          strcpy ( yTitle3, yVal );
          strcpy ( windowTitle3, xVal );
          strcat ( windowTitle3, " vs " );
          strcat ( windowTitle3, yVal );
          break;

        case 4:
```

```
          strcpy ( xTitle4, xVal );
          strcpy ( yTitle4, yVal );
          strcpy ( windowTitle4, xVal );
          strcat ( windowTitle4, " vs " );
          strcat ( windowTitle4, yVal );
          break;

        case 5:
          strcpy ( xTitle5, xVal );
          strcpy ( yTitle5, yVal );
          strcpy ( windowTitle5, xVal );
          strcat ( windowTitle5, " vs " );
          strcat ( windowTitle5, yVal );
          break;

        case 6:
          strcpy ( xTitle6, xVal );
          strcpy ( yTitle6, yVal );
          strcpy ( windowTitle6, xVal );
          strcat ( windowTitle6, " vs " );
          strcat ( windowTitle6, yVal );
          break;

        case 7:
          strcpy ( xTitle7, xVal );
          strcpy ( yTitle7, yVal );
          strcpy ( windowTitle7, xVal );
          strcat ( windowTitle7, " vs " );
          strcat ( windowTitle7, yVal );
          break;

        case 8:
          strcpy ( xTitle8, xVal );
          strcpy ( yTitle8, yVal );
          strcpy ( windowTitle8, xVal );
          strcat ( windowTitle8, " vs " );
          strcat ( windowTitle8, yVal );
          break;

        case 9:
          strcpy ( xTitle9, xVal );
          strcpy ( yTitle9, yVal );
          strcpy ( windowTitle9, xVal );
          strcat ( windowTitle9, " vs " );
          strcat ( windowTitle9, yVal );
          break;

        default:
          break;
      }
    }
    //
    else // dataType = 1: data
    {
      switch ( screenNo )
      {
        case 1:
          dataArray1 [ np1 ].x = ( atof ( xVal ) ) / x1Max;
          dataArray1 [ np1 ].y = ( atof ( yVal ) ) / y1Max;

          if ( n1 < MAXDATA )
          {
            n1++;
          }

          np1 = np1++ % MAXDATA;

          // re-scale occasionally, often enough to keep the points in the visible window
          // but not too frequently to not take up too much processor time (a compromise)
          if ( np1 == SCALE3 || np1 == SCALE1 || np1 == SCALE2 )
```

```
        {
                    x1Max = 1;
                    y1Max = 1;

          // find the max values of x and y in the array
          for ( i = 0; i < n1; i++ )
          {
            x1 = fabs ( dataArray1 [ i ].x );// all values are supposed to be pos, but just in case ↙
not
            y1 = fabs ( dataArray1 [ i ].y );

            if ( x1 > x1Max )
            {
              x1Max = x1;// store the max value of x
            }
            if ( y1 > y1Max )
            {
              y1Max = y1;// store the max value of y
            }
          }

          // normalize x and y to their max values to fit on the points in the visible window
          for ( i = 0; i < n1; i++ )
          {
            dataArray1 [ i ].x = dataArray1 [ i ].x / x1Max;
            dataArray1 [ i ].y = dataArray1 [ i ].y / y1Max;
          }

          // correct for old value previous points were scaled by
          x1Max = oldx1Max * x1Max;
          y1Max = oldy1Max * y1Max;

          // store the current max value in old value for next time scaling occurs
          oldx1Max = x1Max;
          oldy1Max = y1Max;
        }

        break;

      case 2:
        dataArray2 [ np2 ].x = ( atof ( xVal ) ) / x2Max;
        dataArray2 [ np2 ].y = ( atof ( yVal ) ) / y2Max;

        if ( n2 < MAXDATA )
        {
          n2++;
        }

        np2 = np2++ % MAXDATA;

        // re-scale occasionally, often enough to keep the points in the visible window
        // but not too frequently to not take up too much processor time (a compromise)
        if ( np2 == SCALE3 || np2 == SCALE1 || np2 == SCALE2 )
        {
                    x2Max = 1;
                    y2Max = 1;

          // find the max values of x and y in the array
          for ( i = 0; i < n2; i++ )
          {
            x2 = fabs ( dataArray2 [ i ].x );// all values are supposed to be pos, but just in case ↙
not
            y2 = fabs ( dataArray2 [ i ].y );

            if ( x2 > x2Max )
            {
              x2Max = x2;// store the max value of x
            }
            if ( y2 > y2Max )
            {
```

```
            y2Max = y2;// store the max value of y
          }
        }

        // normalize x and y to their max values to fit on the points in the visible window
        for ( i = 0; i < n2; i++ )
        {
          dataArray2 [ i ].x = dataArray2 [ i ].x / x2Max;
          dataArray2 [ i ].y = dataArray2 [ i ].y / y2Max;
        }

        // correct for old value previous points were scaled by
        x2Max = oldx2Max * x2Max;
        y2Max = oldy2Max * y2Max;

        // store the current max value in old value for next time scaling occurs
        oldx2Max = x2Max;
        oldy2Max = y2Max;
      }

      break;

    case 3:
      dataArray3 [ np3 ].x = ( atof ( xVal ) ) / x3Max;
      dataArray3 [ np3 ].y = ( atof ( yVal ) ) / y3Max;

      if ( n3 < MAXDATA )
      {
        n3++;
      }

      np3 = np3++ % MAXDATA;

      // re-scale occasionally, often enough to keep the points in the visible window
      // but not too frequently to not take up too much processor time (a compromise)
      if ( np3 == SCALE3  || np3 == SCALE1 || np3 == SCALE2 )
      {
                  x3Max = 1;
                  y3Max = 1;

        // find the max values of x and y in the array
        for ( i = 0; i < n3; i++ )
        {
          x3 = fabs ( dataArray3 [ i ].x );// all values are supposed to be pos, but just in case ↙
not
          y3 = fabs ( dataArray3 [ i ].y );

          if ( x3 > x3Max )
          {
            x3Max = x3;// store the max value of x
          }
          if ( y3 > y3Max )
          {
            y3Max = y3;// store the max value of y
          }
        }

        // normalize x and y to their max values to fit on the points in the visible window
        for ( i = 0; i < n3; i++ )
        {
          dataArray3 [ i ].x = dataArray3 [ i ].x / x3Max;
          dataArray3 [ i ].y = dataArray3 [ i ].y / y3Max;
        }

        // correct for old value previous points were scaled by
        x3Max = oldx3Max * x3Max;
        y3Max = oldy3Max * y3Max;

        // store the current max value in old value for next time scaling occurs
        oldx3Max = x3Max;
```

```
          oldy3Max = y3Max;
        }

        break;

      case 4:
        dataArray4 [ np4 ].x = ( atof ( xVal ) ) / x4Max;
        dataArray4 [ np4 ].y = ( atof ( yVal ) ) / y4Max;

        if ( n4 < MAXDATA )
        {
          n4++;
        }

        np4 = np4++ % MAXDATA;

        // re-scale occasionally, often enough to keep the points in the visible window
        // but not too frequently to not take up too much processor time (a compromise)
        if ( np4 == SCALE3  || np4 == SCALE1 || np4 == SCALE2 )
        {
                      x4Max = 1;
                      y4Max = 1;

          // find the max values of x and y in the array
          for ( i = 0; i < n4; i++ )
          {
            x4 = fabs ( dataArray4 [ i ].x );// all values are supposed to be pos, but just in case ↙
not
            y4 = fabs ( dataArray4 [ i ].y );

            if ( x4 > x4Max )
            {
              x4Max = x4;// store the max value of x
            }
            if ( y4 > y4Max )
            {
              y4Max = y4;// store the max value of y
            }
          }

          // normalize x and y to their max values to fit on the points in the visible window
          for ( i = 0; i < n4; i++ )
          {
            dataArray4 [ i ].x = dataArray4 [ i ].x / x4Max;
            dataArray4 [ i ].y = dataArray4 [ i ].y / y4Max;
          }

          // correct for old value previous points were scaled by
          x4Max = oldx4Max * x4Max;
          y4Max = oldy4Max * y4Max;

          // store the current max value in old value for next time scaling occurs
          oldx4Max = x4Max;
          oldy4Max = y4Max;
        }

        break;

      case 5:
        dataArray5 [ np5 ].x = ( atof ( xVal ) ) / x5Max;
        dataArray5 [ np5 ].y = ( atof ( yVal ) ) / y5Max;

        if ( n5 < MAXDATA )
        {
          n5++;
        }

        np5 = np5++ % MAXDATA;

        // re-scale occasionally, often enough to keep the points in the visible window
```

```
        // but not too frequently to not take up too much processor time (a compromise)
        if ( np5 == SCALE3 || np5 == SCALE1 || np5 == SCALE2 )
        {
                    x5Max = 1;
                    y5Max = 1;

          // find the max values of x and y in the array
          for ( i = 0; i < n5; i++ )
          {
            x5 = fabs ( dataArray5 [ i ].x );// all values are supposed to be pos, but just in case ↙
not
            y5 = fabs ( dataArray5 [ i ].y );

            if ( x5 > x5Max )
            {
              x5Max = x5;// store the max value of x
            }
            if ( y5 > y5Max )
            {
              y5Max = y5;// store the max value of y
            }
          }

          // normalize x and y to their max values to fit on the points in the visible window
          for ( i = 0; i < n5; i++ )
          {
            dataArray5 [ i ].x = dataArray5 [ i ].x / x5Max;
            dataArray5 [ i ].y = dataArray5 [ i ].y / y5Max;
          }

          // correct for old value previous points were scaled by
          x5Max = oldx5Max * x5Max;
          y5Max = oldy5Max * y5Max;

          // store the current max value in old value for next time scaling occurs
          oldx5Max = x5Max;
          oldy5Max = y5Max;
        }

        break;

      case 6:
        dataArray6 [ np6 ].x = ( atof ( xVal ) ) / x6Max;
        dataArray6 [ np6 ].y = ( atof ( yVal ) ) / y6Max;

        if ( n6 < MAXDATA )
        {
          n6++;
        }

        np6 = np6++ % MAXDATA;

        // re-scale occasionally, often enough to keep the points in the visible window
        // but not too frequently to not take up too much processor time (a compromise)
        if ( np6 == SCALE3 || np6 == SCALE1 || np6 == SCALE2 )
        {
                    x6Max = 1;
                    y6Max = 1;

          // find the max values of x and y in the array
          for ( i = 0; i < n6; i++ )
          {
            x6 = fabs ( dataArray6 [ i ].x );// all values are supposed to be pos, but just in case ↙
not
            y6 = fabs ( dataArray6 [ i ].y );

            if ( x6 > x6Max )
            {
              x6Max = x6;// store the max value of x
            }
```

```
            if ( y6 > y6Max )
            {
              y6Max = y6;// store the max value of y
            }
          }

          // normalize x and y to their max values to fit on the points in the visible window
          for ( i = 0; i < n6; i++ )
          {
            dataArray6 [ i ].x = dataArray6 [ i ].x / x6Max;
            dataArray6 [ i ].y = dataArray6 [ i ].y / y6Max;
          }

          // correct for old value previous points were scaled by
          x6Max = oldx6Max * x6Max;
          y6Max = oldy6Max * y6Max;

          // store the current max value in old value for next time scaling occurs
          oldx6Max = x6Max;
          oldy6Max = y6Max;
        }

        break;

      case 7:
        dataArray7 [ np7 ].x = ( atof ( xVal ) ) / x7Max;
        dataArray7 [ np7 ].y = ( atof ( yVal ) ) / y7Max;

        if ( n7 < MAXDATA )
        {
          n7++;
        }

        np7 = np7++ % MAXDATA;

        // re-scale occasionally, often enough to keep the points in the visible window
        // but not too frequently to not take up too much processor time (a compromise)
        if ( np7 == SCALE3 || np7 == SCALE1 || np7 == SCALE2 )
        {
                    x7Max = 1;
                    y7Max = 1;

          // find the max values of x and y in the array
          for ( i = 0; i < n7; i++ )
          {
            x7 = fabs ( dataArray7 [ i ].x );// all values are supposed to be pos, but just in case ↙
not
            y7 = fabs ( dataArray7 [ i ].y );

            if ( x7 > x7Max )
            {
              x7Max = x7;// store the max value of x
            }
            if ( y7 > y7Max )
            {
              y7Max = y7;// store the max value of y
            }
          }

          // normalize x and y to their max values to fit on the points in the visible window
          for ( i = 0; i < n7; i++ )
          {
            dataArray7 [ i ].x = dataArray7 [ i ].x / x7Max;
            dataArray7 [ i ].y = dataArray7 [ i ].y / y7Max;
          }

          // correct for old value previous points were scaled by
          x7Max = oldx7Max * x7Max;
          y7Max = oldy7Max * y7Max;
```

```
        // store the current max value in old value for next time scaling occurs
        oldx7Max = x7Max;
        oldy7Max = y7Max;
      }

      break;

    case 8:
      dataArray8 [ np8 ].x = ( atof ( xVal ) ) / x8Max;
      dataArray8 [ np8 ].y = ( atof ( yVal ) ) / y8Max;

      if ( n8 < MAXDATA )
      {
        n8++;
      }

      np8 = np8++ % MAXDATA;

      // re-scale occasionally, often enough to keep the points in the visible window
      // but not too frequently to not take up too much processor time (a compromise)
      if ( np8 == SCALE3 || np8 == SCALE1 || np8 == SCALE2 )
      {
                    x8Max = 1;
                    y8Max = 1;

        // find the max values of x and y in the array
        for ( i = 0; i < n8; i++ )
        {
          x8 = fabs ( dataArray8 [ i ].x );// all values are supposed to be pos, but just in case ↙
not
          y8 = fabs ( dataArray8 [ i ].y );

          if ( x8 > x8Max )
          {
            x8Max = x8;// store the max value of x
          }
          if ( y8 > y8Max )
          {
            y8Max = y8;// store the max value of y
          }
        }

        // normalize x and y to their max values to fit on the points in the visible window
        for ( i = 0; i < n8; i++ )
        {
          dataArray8 [ i ].x = dataArray8 [ i ].x / x8Max;
          dataArray8 [ i ].y = dataArray8 [ i ].y / y8Max;
        }

        // correct for old value previous points were scaled by
        x8Max = oldx8Max * x8Max;
        y8Max = oldy8Max * y8Max;

        // store the current max value in old value for next time scaling occurs
        oldx8Max = x8Max;
        oldy8Max = y8Max;
      }

      break;

    case 9:
      dataArray9 [ np9 ].x = ( atof ( xVal ) ) / x9Max;
      dataArray9 [ np9 ].y = ( atof ( yVal ) ) / y9Max;

      if ( n9 < MAXDATA )
      {
        n9++;
      }

      np9 = np9++ % MAXDATA;
```

```
          // re-scale occasionally, often enough to keep the points in the visible window
          // but not too frequently to not take up too much processor time (a compromise)
          if ( np9 == SCALE3 || np9 == SCALE1 || np9 == SCALE2 )
          {
                        x9Max = 1;
                        y9Max = 1;

            // find the max values of x and y in the array
            for ( i = 0; i < n9; i++ )
            {
              x9 = fabs ( dataArray9 [ i ].x );// all values are supposed to be pos, but just in case ↙
  not
              y9 = fabs ( dataArray9 [ i ].y );

              if ( x9 > x9Max )
              {
                x9Max = x9;// store the max value of x
              }
              if ( y9 > y9Max )
              {
                y9Max = y9;// store the max value of y
              }
            }

            // normalize x and y to their max values to fit on the points in the visible window
            for ( i = 0; i < n9; i++ )
            {
              dataArray9 [ i ].x = dataArray9 [ i ].x / x9Max;
              dataArray9 [ i ].y = dataArray9 [ i ].y / y9Max;
            }

            // correct for old value previous points were scaled by
            x9Max = oldx9Max * x9Max;
            y9Max = oldy9Max * y9Max;

            // store the current max value in old value for next time scaling occurs
            oldx9Max = x9Max;
            oldy9Max = y9Max;
          }

          break;

        default:
          break;
      }
    }// else dataType = 1: data

    // after values have been entered and normalized, send an ack "x" to the client so they send the ↙
  next data
    if (send ( clientSock, "x", 2, 0) == SOCKET_ERROR )
    {
      itoa (WSAGetLastError (), error, 10);
      MessageBox (NULL, error, "Error, sending data to the client failed", MB_SETFOREGROUND );
    }
  }//while

  // Disable both sending and receiving on clientSock.
  shutdown ( clientSock, 0x02 );

  // Close clientSock.
  closesocket ( clientSock );

  WSACleanup ();

  return 0;
}
```