```
////////////////////////////////////////////////////////////////////////
//  CS75202 Computer Communication Networks
//
//  File:     visClient.cpp
//  Author:   Kenneth W Schmidt
//  Abstract: This program is the driver for vis.cpp.  It should be incorporated
//            in the program that generates the statistics for the router.
//            As shown below, it connects to vis.cpp on port 32767, an
//            unassigned IANA port that the listening display listens on,
//            and asks for an input of hostName which can be either
//            a resolvable name like "localhost" or "b1" or it can be
//            the IP address of the host running the listener and statistics
//            display (this can be changed to hard-coded if desired).
//            The code below also shows a driver which just reads in a file
//            for testing purposes, remove this section when incorporating
//            in the statistics generator.  Data is in the following format:
//            "dataType screenNo xData yData".  All data is separated by spaces.
//            dataType can be 0 (setup, send the name of the x axis and y axis)
//            or 1 ( data, values of x and y data ).  For example:
//            "0 1 Throughput Delay" tells the listening display server that
//            this is setup data for screen 1, the x axis lable is Throughput
//            and the y axis label is Delay.  Next example:
//            "1 1 0.56 1.42" tells the listening display server that
//            this is data for screen 1, the x axis value is 0.56 and the
//            y axis value is 1.42.  Build strings to this format before sending
//            them to the listening display server (vis.cpp)
//  Revision History:   Date          Who      Description
//              ------------------------------------------
//                      4/4/04        KWS      Initial Release
////////////////////////////////////////////////////////////////////////


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

#include <winsock.h>
#include <iostream.h>
#include <fstream.h>    // to read from a file
#include <stdio.h>

#define MAXNAME 50                    // max length of the host name or IP address
#define MAXSTRING 1000                // max string length to send to the listening data display
#define PORT 32767                    // unassigned IANA port no (used by display listener)

int main()
{
  int serverSocRetVal = 0;            // Return value of recv function
  char clientData [ MAXSTRING ] = "";  // data to be sent from client to display listener
  char hostName [ MAXNAME ] = "";     // name or IP address of the host running the display listener

                                      // an ack from the display listener that it is ready to
  char receivedData[1];               // receive more data, must be a char* type

                                      // Initilaize socket to invalid so we can check to
  SOCKET serverSock = INVALID_SOCKET;  // see if a valid socket connection has been made
  SOCKADDR_IN serverSin;              // Address of the server socket

                                      // A pointer to the HOSTENT struct of the server
  PHOSTENT phostent = NULL;           // containing name, address, address length (& other data)
  WSADATA WSAData;                    // A struct of winsock data for WS2-32.dll

  // get the name or IP address of the host running the display listener
  cout << "Enter the name or IP address of the host running the display listener" << endl;
  cin >> hostName;

  // Initialize winSock
  if (WSAStartup (MAKEWORD(1,1), &WSAData) != 0)
  {
    cout << "Error, winSock failed to initialize: " << WSAGetLastError () << endl;
    return FALSE;
```

```
  }

  // Create a TCP/IP socket
  if ((serverSock = socket (AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
  {
    cout << "Error, allocating the socket: " << WSAGetLastError () << endl;
    return FALSE;
  }

  // Fill out the server socket's address information.
  serverSin.sin_family = AF_INET;

  // Retrieve the host information corresponding to the host name then close the socket
  if ((phostent = gethostbyname (hostName)) == NULL)
  {
    cout << "Error, unable to read host name: " << WSAGetLastError () << endl;
    closesocket (serverSock);
    return FALSE;
  }

  // Assign IP address to the socket (copied from hostName data)
  memcpy ((char FAR *)&(serverSin.sin_addr),
      phostent->h_addr,
      phostent->h_length);

  // Convert from host byte order to network byte order (big-endian)
  serverSin.sin_port = htons ( PORT );

  // Establish a connection to the server socket.
  if (connect (serverSock, (PSOCKADDR) &serverSin, sizeof (serverSin)) == SOCKET_ERROR)
  {
    cout << "Error, connecting to the server failed: " << WSAGetLastError () << endl;

    // Close socket on error
    closesocket (serverSock);
    return FALSE;
  }

//*******************************************************************************
  // modify this section to send data from the statistics generator rather than
  // reading from a file that is shown below.

  char inputTitle [ MAXNAME ];// name of input file to read
  ifstream readFile;              // to read from a file

  while (true)
  {
    // Get the data to be sent
    cout << "Enter the name of the file to be sent to the display listener" << endl;
    cin >> inputTitle;
    cout << endl;

    readFile.open ( inputTitle, ios::in );// open the file to be read

    if ( !readFile )// did the input file open successfuly
    {
      cout << "can't open the file to be read from" << endl;
    }

    while ( !readFile.eof() ) // read the data from the file to the end of the file
    {

      readFile >> clientData;// get data from the file to send to the listening server

      // Send the data to the display listener.
      if (send (serverSock, clientData, strlen (clientData) + 1, 0) == SOCKET_ERROR)
      {
        cout << "Error sending data to server: " << WSAGetLastError () << endl;
      }
```

```
      // Receive ack from the display listener
      serverSocRetVal = recv (serverSock, receivedData, 1024, 0);

      // Check if there is any data received from the server
      // If there was an error creating a socket connection
      if (serverSocRetVal == SOCKET_ERROR)
      {
        cout << "Error, connecting to server: " << WSAGetLastError () << endl;
        break;
      }

      // If the connection is OK but no data received
      else if (serverSocRetVal == 0)
      {
        cout << "End of Data return value " <<  WSAGetLastError () << endl;
        break;
      }
      // else Connection is OK and data received from server OK, so continue sending next data
      // don't need to do anything with the actual data that the display listener sent
      //cout << receivedData << endl;
    }
    readFile.close();
  }
//*******************************************************************************************

  // Shutdown the socket
  shutdown (serverSock, 0x00);
  shutdown (serverSock, 0x01);

  // Close the socket
  closesocket (serverSock);

  WSACleanup ();

  return 0;
}
```