Multimedia Programming 2004/2005

Assignments No. 5

January 19th 2005

Assignment 1 due: January 26th 2005

Goals of the assignments:

- Learn how to use CList and CString classes in C++
- Learn how to send and receive messages over a TCP/IP network
- Learn how to use client/server programming

Preparations:

1. Download the code (code05a.zip (assignment 1) and code05b.zip (the rest of the assignments)) from the MMP2004 web-site and unzip it to a local directory
2. All further directories mentioned in the assignments can be found in this local directory

Posting Your Work: See the last page of Assignment Set 1 for the procedure for posting your work. NB Only assignment 1 has to be posted.

Assignment 1: Classes CList and CString

In Code05a.zip several examples can be found on how to use the MFC-classes *CList* and *CString*. Compile and study the code. The *CDialogExample* shows how the *CList* class can be used to store an arbitrary long list of pointers to the class *CPerson*. In *CPerson* personal data is stored:

- Extend the program *CDialogExample* such that also the address, age, and gender of the person can be stored and shown in the dialog when browsing through the list. (Add these data for at least 8 persons in the initialization phase.)
- Add some useful comment that explains the working of the program.
- There are several semantic errors in the program: 1) the first element of the list is never shown; 2) if you browse passed the head or tail, you can not go back to the next, or previous, respectively. Fix these semantic errors.

Assignment 2: Sockets

1. Browse to the *Sockets* directory. There you will find two directories. The *Server* directory contains the socket server code. The socket server listens on a socket for client requests. The directory *Client* contains the socket client software that can make a socket connection to a running socket server. First, go to the *Server* directory and compile and execute the server code. Then go to the *Client* directory, set the program arguments to the address of the machine that runs the server (<Project><Settings><Debug><Program Arguments>). Compile and run the client program.
2. Change the server program so that every client that issues a request to the server gets a different response.
3. Just try this part for your self: look at the comment in the server program about the *Dummy HTML server*. Uncomment that code and be sure that the default welcome

message is not sent. Try to use a Netscape browser to get and show the page (use http://machineadress:2222 , where *machineaddres* is equal to the IP-address or the name of the machine on which the server runs, and *2222* is the port used in the server program).

Assignment 3: Client/Server using DirectPlay

1. Browse to the *SimpleClientServer* directory and get the *SimpleServer* code to compile. Run the *SimpleServer* code and press the *Start Server* button. This starts a *game* server that will listen to a specified TCP/IP port for players to enter the game. Furthermore, it will process all messages of players that entered the game. Keep the server running. Note that you can have several *game* servers running at the same time (listening on different ports).
2. Now go to the SimpleClient subdirectory and build the executable. Run the code at least two times. Enter different player names and press the *Start Search* button on each *SimpleClient* dialog. After it found a game press the *Join* button. By pressing the *Info* button you can see who joined the game.
3. Try to connect to the game servers of other students.
4. Describe in terms of messages (sending, receiving, and processing) what happens if a player presses the *Wave to other players!* button.

Assignment 4: Peer to Peer using DirectPlay

1. Browse to the *SimplePeer* directory and get the *SimplePeer* code to compile. Run the *SimpleServer* code with connection type *DirectX8.0 TCP/IP Service Provider*. Press the *Create* button to create a game. Again this starts a *game* server that will listen to a specified TCP/IP port for players to enter the game.
2. Execute *SimplePeer* once more with connection type *DirectX8.0 TCP/IP Service Provider*. Press the *Start Search* button and fill in *127.0.0.1* as hostname. (Note *127.0.0.1* is the IP address of the *localhost*.) Join the game that is found. Execute several different *SimplePeers* this way. Wave to other players.
3. Try to connect to the *SimplePeer* game servers of other students.
4. Describe in terms of messages (sending, receiving, and processing) what happens if a player presses the *Wave to other players!* button.

Assignment 5: Chat using Peer to Peer

1. Browse to the *ChatPeer* directory and build and run the *ChatPeer* code. Try to start one game host and several clients to that game.
2. Try to connect to *ChatPeer* hosts of other students.
3. Describe in terms of messages (sending, receiving, and processing) what happens if you press the Send button after you entered some text in the dialog edit box.

Assignment 6: Virtual Chat using Peer to Peer

1. Browse to the *Babble* directory. Here you will find an *Eliza* program. Build and execute the code. In the file *babble.txt* all the keywords and possible responses are given. Add your own associations, or make a complete new *Eliza-bot* association file.
2. The class *CEliza* can easily be re-used in other programs to add automatic responses on input text. Add the *CEliza* class to the *ChatPeer* project of Assignment 4 such that the *ChatPeer* server will automatically respond to chat-text of other players in the game. That is, if a user '*Eliza*' creates a game, the player/creator '*Eliza*' will respond automatically on chat messages: if a player *Itisme* joins the game and enters some text in the chat box '*Eliza*' will automatically generate answers to that text using the *Eliza* class.