Jacob Malcy
TA: Vipra Gupta
Section: 104
Assignment 8 Project Proposal

What is your project?
My project is a simple, yet somewhat secure, person to person messaging system that works over your Local Area Network (LAN). It will also include a sort of chat bot that can add things to the chat or help you with things (later described). At the moment, this will only work as a 2 person chat room, no more than 2 people can talk. Whether this will not be the final case, I am not sure at the moment. (Note: I have already done a proof of concept (POC) for this project, I know I can do it.)(Secondary Note: I realize I may have to do multi-threading in order for this to work. Please don't try to stop me. I've done it in Java and I can figure out how to do it in C++ as well!)

What is the goal of this project?
The goal of this project is to create a functional, easy to use chatting system that can be run and shut down efficiently.

Attach your complete .h files with comments to provide some basic idea of the functions.
networkManager.h:

```
#ifndef NETWORKMANAGER_H
#define NETWORKMANAGER_H
#include <iostream>
#include <fstream>
#include <encodedMessage.h>
#include <vector>

using namespace std;

class networkManager
{
    public:
        networkManager();       // Default constructor
        networkManager(bool, string); // Constructor to set if you're host, and
the destination or source IP address
        networkManager(bool, string, string); // Constructor that sets if you're
host, the destination IP or source IP address, and user name
        ~networkManager();

        bool amIHost();         // returns true if you're hosting, false otherwise
        string getDest();       // Get the destination IP address
        string getSource();     // Get the source IP address
        string getName();       // Get your user name
        bool connectionOpen();  // Tell you if the connection is open or not.

        int connect();          // The method that initiates the connection.
        int closeConnection();

        bool setDest(string);   // Change the destination IP address
        bool setSource(string); // Change the source IP address
        bool setName(string);   // Change your user name
        bool setMode(bool);     // Change from server mode to client mode.

        bool sendMessage(encodedMessage);
```

```
    private:
        bool openConnection;    // Variable to know if the connection is open or
not.
        bool isHost;            // Variable to know if you're in server or client
mode.
        string destinationIP;   // Variable to keep track of where you're sending
messages to.
        string sourceIP;        // Variable to keep track of your source IP. Seems
unimportant but will be used in the code.
        string userName;        // Sets the user name that will appear in the chat
as you talk.
        vector <encodedMessage> messagesReceived;
};

#endif // NETWORKMANAGER_H
```

networkFluffer.h: (aka the chat bot)

```
#ifndef NETWORKFLUFFER_H
#define NETWORKFLUFFER_H
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;


class networkFluffer
{
   public:
      // Constructors
      networkFluffer();                 // Default constructor
      networkFluffer(string);             // Constructor that sets the name of the network fluffer
      networkFluffer(string, char, char);     // Constructor that sets the name, activator, and markup
character of the fluffer.

      ~networkFluffer();                 // Obligatory destructor

      // Getters
      string getName();
      char getActiv();
      char getMark();
      void getInsults(vector <string>);
      void getCompliments(vector <string>);

      string markup(string);              // Convert a string to a marked up string should it have the
markup character. If it doesn't have the character return the original string.
      void command(string);                // Do a command if the beginning of the line starts with the
activator character. act accordingly if it has a valid command

      // Setters
      bool setName(string);
      bool setActivator(char);
      bool setMarkup(char);
```

```
        // Add insults or compliments, respectively
        bool addInsults(vector <string>);
        bool addCompliments(vector <string>);


        bool eraseInsults();                // Wipe the insult vector.
        bool eraseCompliments();            // Wipe the compliment vector.
        string removeCompliment(int);       // Removes an compliment at provided index from the
compliment array
        string removeInsult(int);           // Removes an insult at provided index from the insult array

    private:
        // Commands
        string roll(string);                // Roll a N sided dice X times. String entered as XdN
        string insult(string, int);         // Grabs a specified number of random insults from the insults
array and returns them
        string compliment(string, int);     // Grabs a specified number of random compliments from
the compliments array and returns them
        string yell(string);                // Makes a lot of spaces and essentially shouts something for
you
        void tts(string);                   // Text to speech? Not sure if i want to do this.

        // Data members
        string name;                        // Name of the fluffer. Personally, I like "Choity" the chat bot :D
        char activator;                     // The character that will used for activation. Can't be a letter.
        char markupChar;                    // The character that is used in markup, default '*'
        vector <string> insults;            // Vector array of insults
        vector <string> compliments;        // Vector array of compliments
};

#endif // NETWORKFLUFFER_H
```

encodedMessage.h:

```
#ifndef ENCODEDMESSAGE_H
#define ENCODEDMESSAGE_H
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;


class encodedMessage
{
    public:
        // Constructors
        encodedMessage();                   // Default constructor
        encodedMessage(string, bool);       // Constructor that allows you to
specify what the original message is, or what the encoded message is, depending on
the boolean variable.
        encodedMessage(string, string);     // Constructor that allows you to
specify the secret message and the algorithm to use.

        // Destructor
        ~encodedMessage();                  // Obligatory destructor

        // Setters
```

```
        bool setMessage(string);            // Set the message. Wont allow you to
do so if it has been sent.
        bool setAlgo(string);               // Set the algorithm for the encoding.
List of valid encoding options will be provided later.
        bool send();                        // Set the sent variable to true.
Cannot be set back to false.

        // Getters
        string getMessage();                // Get the message that has been set.
        string getAlgo();                   // Get the algorithm that will be used
for encoding.
        bool beenSent();                    // Returns true if the message has
been sent, false otherwise.

    private:
        bool encode();                      // This will be called to do the
encoding on the message.
        bool decode();                      // This will be called to decode a
message.
        string message;                     // Data member to store the unencoded
message. Will be erased upon encoding.
        string secretMessage;               // Data member to store the encoded
message.
        string algo;                        // Data member to store

        bool encoded;                       // Data member to know if the message
has been encoded
        bool messageSet;                    // Data member to know if the encoded
message has been set.
        bool sent;                          // Data member to know if the encoded
message has been sent.
};

#endif // ENCODEDMESSAGE_H
```

Explain why you designed your class this way.

networkManager.h:

The network manager class is designed to be able to host a chat server or connect to a chat server quickly and easily. It will take care of the sending and receiving of data to and from the other person. I have set a boolean variable to know if you're the host or not, along with 3 string variables to know the destination IP address (basically the place where you direct your messages to), source IP address (always your address. Not sure if I want the user to input this or to grab it myself.), and one to keep track of your user name that will appear in the chat. There is also a vector array of encoded messages that have been received.

encodedMessage.h:

(**NOTE**: This protection will keep you protected against script kiddies and your average friend. It will NOT protect you against state actors, CU computer science, cryptography, or ITP faculty, or anyone looking over your shoulder. This is **encoding**, not **encryption**.)

This is my class that allows for some extra security. This class will take in a basic string of characters and encode or decode them with one or more encoding algorithms (ROT-13, base64, base8, and some others that I'll come up with.) It does not allow you to send the message after the 'sent' boolean variable is set to true.

networkFluffer.h:

This class is what handles all of the in-chat commands and markup requests. In chat commands will be given by the activator character, which is by default '!'. Putting this at the beginning of the chat allows you to call the roll, insult, compliment, yell, or Text-to-Speech (maybe?) commands. There will also be a markup character to enable **bolding**, *italicization*, underlining, ~~and maybe strike-through~~. How many of the markupChar that you put before and after your selected text determines which markups to use. By default, markupChar = '*'.

Why have you chosen these data members and how do you plan to use them?

networkManager.h:

- bool openConnection
  - A boolean variable to know if the connection is open. Will be used to stop or allow the sending and receiving of data.
- bool isHost
  - A boolean variable to know if you're the host/server or the client.
  - Will be used to determine how data is sent and/or received.
- string destinationIP
  - Will be used to determine where messages must be sent.
- string sourceIP
  - Will be used to setup server if needed.
- string userName
  - Will be outputted to everyone when a message from that person is sent/received
- vector <encodedMessage> messagesReceived
  - Will be used to keep track of who said what and when.
  - Will also be sent to a log file line by line once the chat is over.

encodedMessage.h:

- string message
  - The plaintext message that will be sent once it is encoded.
- string secretMessage
  - The message that will actually be sent, which is the encoded version of 'message'.
- string algo
  - The variable that determines the algorithm used for encoding. May actually be in a list format seperated by commas, not sure yet.
- bool encoded
  - A variable to check if the message has been encoded yet.
- bool messageSet
  - checks if the non encoded message has been set yet.
- bool sent
  - A variable to know if the message has been sent yet. If it has, don't allow any more changes.

networkFluffer.h:

- string name
  - The name of the networkFluffer/chatBot
- char activator
  - The character that will initiate a command in the chat.
  - Default: !

- char markupChar
  - The character that will markup your text.
  - Default: *
- vector <string> insults
  - A list of insults that can be called randomly
  - Defaults will be developed.
- vector <string> compliments
  - A list of compliments that can be called randomly
  - Defaults will be developed.

Explain how the design meets the requirement.

I've got more than two classes (networkFluffer, networkManager, encodedMessage), each one has at least 4 data members, and they will all have their appropriate getters and setters. File IO will be implemented in that the networkManager will be able to read in files and write to them as chat logs, using a while/for loop. There will be plenty of if/ if-else statements in the network fluffer (such as with the command and markup methods). The compliments and insults methods will allow you to specify the number of insults, using loops.