EECS 211 Assignment 3
Due: Thursday, April 28, 2016

In this assignment we begin to develop classes for our project. Specifically, this assignment implements a class, **datagram**, which will hold information in messages that are transmitted around our simulated network. We also develop a class, **IPAddress**, the form of Internet addresses.

In this assignment we create our first multi-file project for this term. Your project will have six files, although most of them will have only a few lines. Only three files will be of non-trivial length. During the remainder of this project we will add to these files and add additional files.

Background:

Messages transmitted around the Internet contain much more information than just the data that is entered or read by the user. For example, each message contains at least the addresses of the source computer and destination computer as well as the "data" itself. Most messages contain much more information, such as time stamps, byte counts, etc. Each of the various protocols, such as IP and TCP/IP, has a specific format for the messages. In our project, a message will be called a **datagram** and will contain source and destination addresses, the number of bytes in the data part of the datagram, and the actual data.

Internet addresses, or IP addresses, are written in the form $n_1.n_2.n_3.n_4$, where each n_i is an unsigned integer in the range 0-255. Thus, each portion of an IP address corresponds to one byte. Thus, each IP address is stored as four bytes and written/displayed as four non-negative integers separated by dots.

In this assignment we will develop two classes – one for IP address and one for datagram. These will be used throughout the remainder of this project.

We will also introduce the notion of validity checking for data and the use of error codes as return values for functions. In robust systems and applications it is necessary for upper level functions to know whether lower level functions have succeeded or not. In case a lower level function fails to perform its function, the upper level function must take some corrective action. A simple example familiar to most students is an ATM. If the cash dispenser function fails to dispense the cash, the ATM should take some corrective action, like display a message to the user and send a message to the maintenance department. You are also already using this because your own main function in your projects returns 0 when it's done. By default, 0 is used to indicate success, and non-zero return values indicate errors. Most systems define a large number of error codes so that top level portions of the system can recover properly depending on what the error was.

Assignment:

Build a project with the following six files:

definitions.h	**main.cpp**
system_utilities.h	system_utilities.cpp
datagram.h	**datagram.cpp**

These files are provided for you with some base code. The bolded names are the only ones you need to alter. Do not alter any of the others.

definitions.h

Currently, this file contains a constant called BAD_IP_ADDRESS, which is set to 200.
You don't need to do anything to this file.

datagram.h and datagram.cpp

datagram.h contains the interfaces for two classes, IPAddress and datagram. You don't need to do anything to this file. In datagram.cpp, you need to implement the functions that are shown (prototyped) in datagram.h.

Here is a brief overview of these two classes, including descriptions of them member functions that you need to define:

Class **IPAddress**:

- data members: an array of 4 **unsigned ints**.
- member functions:
    - **int parse(string s)** – The argument is a string representing an IP address in the format described above (four numbers with dots between). Extract the four numbers from the string and assign to the four data members. You may assume the string has the proper format (sequence of digits and dots). However, you must check that all the numbers are in the range 0-255. If so, return 0; otherwise, set all four data members to 0 and return BAD_IP_ADDRESS.
    - **void display()** – Print the IP address in the format described above. Do not include any other characters, such as line feed ('\n' or endl) or spaces, other than the four numbers and the three dots.
    - **int sameAddress(IPAddress x)** – Return 1 if the IPAddress x is the same (i.e., all four elements are equal respectively) as the invoking IPAddress object, 0 otherwise.

Class **datagram**:

- data members:
    - two IPAddress objects representing source and destination IP addresses.
    - an integer representing the number of bytes in the data part of the datagram.
    - a string representing the message part of the datagram.
- member functions:
    - **void makeDatagram(IPAddress s, IPAddress d, string m)** – The arguments s and d are the source and destination IP addresses, respectively. The argument m

is a string of the message itself. Use these arguments to fill in the data members of the datagram. Don't forget about the length data member, which should be set to be the length of the message.

- **void display(void)** – Print the datagram in the following format:
  SOURCE: *source IP address*
  DESTINATION: *destination IP address*
  MESSAGE LENGTH: *number of bytes*
  MESSAGE: "*message*"
  blank line

  Note: There is a single space between the colon and the item to the right. There are double quotes around the message itself.

system_utilities.h and system_utilities.cpp

systems_utilities.h contains prototypes for two functions. You don't need to do anything to this file. In system_utilities.cpp, these two functions are defined for you. You also don't need to do anything to this file.

- **void printError(int errcode)** - The argument is an error code. This function should print an appropriate message depending on the argument. At the moment we have only one error code, but in future assignments we will add more.
- **void wait()** - This function has no arguments. It prompts the user to press enter and waits for the press. This is useful when displaying large amounts of data on the screen; it prevents the program from moving on until the user (you or the TAs) have had a chance to examine the output.

main.cpp

Use main.ccp to hold a main function that tests your classes. Below, I've outlined my suggestion for a good approach to testing. Note, I will not be grading your main function, so the following steps are included merely to help you test your classes.

Your main function should include at least the following variables:

- an integer variable, **error_code**, to hold error codes
- an array of 10 elements of type **IPAddress**
- a variable of type **datagram**

The main function should do the following:

- Have the **IPAddress** elements parse the following ten strings:
  "1.2.3.4"
  "255.255.255.0"
  "0.0.0.255"
  "1.2.3.4"
  "255.255.255.0"
  "0.0.0.0"
  "256.1.1.1"
  "0.300.378.1"
  "1.2.3.512"

"2.2.2.4"

For each parse assign the return value to **error_code**. If a parse returns BAD_IP_ADDRESS, call **printError** with **error_code** as the argument. Your first **IPAddress** element should parse the first string and then display itself, and then you should check error_code. Then your second **IPAddress** element should parse the second string and display itself, after which you check error_code, and so on for all 10 elements.

- Call the **wait** function to hold the output on the screen until the user presses ENTER.
- Compare each pair of **IPAddress** objects in your array (compare element 0 with elements 1, 2, … 9, then element 1 with elements 2, 3, … 9, etc.) For each pair that are the same, print the two indices with a message saying those two are the same IP address.
- Have the **datagram** variable invoke its **makeDatagram** method using the first two IP addresses and the message "Datagram 1". Then have the datagram display itself. Do the same with the third and fourth IP addresses and the message "This is a test." and again with the fifth and sixth IP addresses and the message "Dear Mom and Dad:".

Grading

As noted above, I will not be testing the output of your main function, rather I will write my own main function that creates instances of the IPAddress and datagram classes and tests the member functions independently. That said, the display functions for those two classes should follow my instructions, as I will be checking that the outputs are exactly correct.

Checking your work on the EECS Servers

Before turning in your project, its critical that you make sure that your code compiles and runs on the EECS servers. Here's how:

1) Copy your code (all 6 code files + the provided makefile) to the eecs server using scp or pscp
2) Connect to the servers (ssh or putty) and in the same directory as the 7 copied files, type 'make' to build your code.
3) Type ./Assignment3 to run your code. Verify that the output is as you expect.

What To Turn In

Since you only modified two files, and I just said that one of them will not be graded, you only need to submit one file (datagram.cpp). Before uploading this file to canvas, first rename the file as YOUR_NET_ID.cpp as you have for the last two assignments. Upload ONLY THIS FILE on canvas.

Hints:

The **parse** function needs to convert sequences of characters into integers. Remember the character form of a digit is not the same internally as the numeric value of the digit. For example, the digit 0 is stored internally as the byte 30_{HEX} (or, 48_{DEC}). A string representing an integer is stored as a sequence of digit characters, not a four-byte normal integer. So, for example, the second IP address mentioned above is "255.255.255.0", which is the character string

32_{HEX} 35_{HEX} 35_{HEX} $2E_{HEX}$ 32_{HEX} 35_{HEX} 35_{HEX} $2E_{HEX}$
32_{HEX} 35_{HEX} 35_{HEX} $2E_{HEX}$ 30_{HEX} 0

Your parse function will need a loop to read individual characters until a dot or end of string (0) is reached. For each digit it encounters, it must convert from character form to numeric form (by subtracting 48_{DEC} or, equivalently '0') and then multiply the value previously collected by ten and add the new digit.