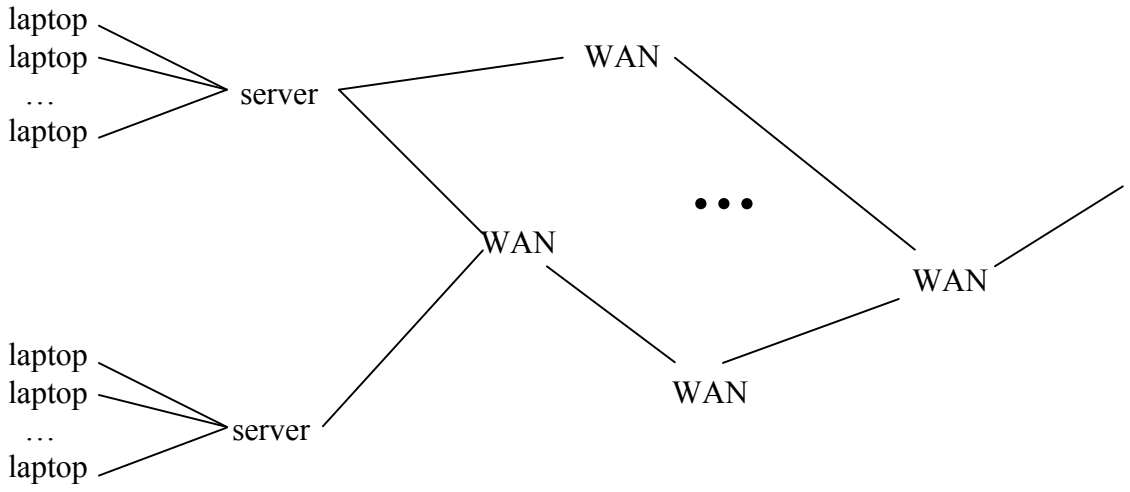
EECS 211 Assignment 6
Due: Friday, May 20, 2016

In this assignment we develop the general class for network node and three derived classes that represent different kinds of computers that will form our simulated network.

Background:

The internet includes a multitude of computers connected together in a network. Individual computers in the network are called nodes, and each node has its own IP address. Computers can attach to the network at any time, for example when you turn your laptop's wireless on. Computers in the network can drop out of the network, for example when you turn your computer off. Within the network there are different kinds of computers. For example, your laptop may connect to a local area network server when you turn your wireless card on; the local server may have some special capabilities, such as URL look-up and storage for internet messages from many laptops. A local server may connect to a special computer, called a domain-name server (DNS), to check the URLs that you type into your browser. Of course, any given computer in the network is not connected to every other computer, only to a small number of computers. Your laptop, for example, may only be connected to your local server; your local server may only connect to one or two nearby DNS nodes as well as to a limited number of nearby other servers. When data is transmitted from one computer to another, it actually travels through possibly many intermediate nodes in the network.

In our project we will simulate a simplified computer network. There will be three kinds of nodes. Laptop computers will generate datagrams. They will also be the end target for datagrams generated by other laptops in the network. That is, a datagram in our network will be sent from one laptop to another laptop. A server computer connects laptops to the rest of the network. Each server receives and holds datagrams from its local network for transmission to other computers. Each server is connected to one or more network servers, called WAN (Wide Area Network) computers in our project. The WAN computers are connected to each other and facilitate transmission of datagrams across the network. Pictorially,



Class Structure:

In this assignment we will make several classes for the various kinds of machines that will make up the network. There are three kinds:

> **laptop** – A laptop can generate and receive datagrams. Each laptop is connected to a single server (local area network) host.
>
> **server** – A server can be attached to several laptops and also to one or more WAN machines. A server can send/receive datagrams to/from any computer to which it is connected. A server cannot generate datagrams on its own.
>
> **WAN** (Wide Area Network machine) – A WAN can be attached only to server and WAN machines. A WAN can send/receive datagrams to/from any computer to which it is connected. A WAN cannot generate datagrams on its own.

We will develop a class, called **node**, which will be used to derive the above three classes. The **node** class will hold data, such as IP address, and provide functions that are common to all computers in our simulation. Each derived class will then add its own data and function members appropriate for that class. Each of the classes **laptop**, **server**, and **WAN** is publicly derived from **node**. Notice that the definitions of these four classes are provided in machines.h.

Each **server** and each **WAN** has two queues for holding datagrams – one for holding incoming datagrams, and one for holding outgoing datagrams. The size of these queues is determined when the computer is created. Each **laptop** can hold one incoming datagram and one outgoing datagram.

Each **server** can be connected to as many as eight **laptop** machines and as many as four **WAN**s. Each **WAN** can be connected to as many as four **server**s and as many as four other **WAN**s.

New Constants:

Notice that the definitions of the following constants have been added to your definitions header file:

LAPTOP	1
SERVER	2
WAN_MACHINE	3
UNKNOWN_MACHINE_TYPE	201
NETWORK_FULL	202
NO_SUCH_MACHINE	203
MAX_MACHINES	5

Assignment:

Part1: Classes in machines.h/.cpp
Below is a description of the four new classes. Looking at the fnction prototypes from the machines.h file, your job is to implement the reamining member functions in the machines.cpp file. Notice that a few are provided for you in machines.cpp. Also notice an expanded printError function system_utilities.cpp.

Class **node**:

- data members:
  - a pointer to a string called **name** holding the name of this computer.
  - **my_address**, the IPAddress of this node in the network.
- function members:
  - node(string n, IPAddress a) – an initializing constructor that receives a string and an IP address as arguments. The constructor sets the node's IP address data member to a. The constructor allocates new space for the name and copies the string pointed to by n.
  - ~node() – free the space used for the name.
  - virtual void display() – a function that displays the computer's name and IP address in suitable and informative format.
  - int amIThisComputer(IPAddress x) – returns 1 if the node's own IP address is the same as x, 0 otherwise.
  - myType will be a member function of all four classes. It simply returns the numerical constant representing the type of machine. For example, LAPTOP. See definitions.h for other constants representing machines. Notice that node does not have a numerical constant. Since we will not be creating node machines (we'll be creating more specifically laptops, servers, etc), the myType function in the node class should just return 0.

Class **laptop**:

- data members in addition to those already in **node**:
  - two datagram pointers – one for an incoming datagram and one for an outgoing datagram.
  - an IP address – the IP address of the server to which the laptop will be connected.
- function members:
  - laptop(string n, IPAddress a) – an initializing constructor. This function calls the parent class constructor to initialize the computer name and IP address. It then sets the two datagram pointers to NULL. The IP address of the server should be set to "0.0.0.0"; this address indicates no connection. Connections within the network will be implemented in Programming Assignment 7.
  - void display() – a function that displays the information in this object. This function prints the text "Laptop computer:" and then calls the parent class display function. It then displays the incoming and outgoing datagrams with suitable headings. For each datagram pointer if the pointer is not NULL (i.e., it's pointing to an actual datagram) display the datagram, otherwise display "No datagram."
  - void initiateDatagram(datagram* d) – a function that sets the outgoing datagram

pointer to message.
    - void receiveDatagram(datagram* d) – a function that sets the incoming datagram pointer to message.

Class **server**:

- data members in addition to those already in **node**:
    - an array of 8 IP addresses representing connections to laptop machines.
    - an array of 4 IP addresses representing connections to WAN machines.
    - two integers
- function members:
    - server(string n, IPAddress a) – an initializing constructor. This function calls the parent class constructor. The constructor also sets the two integer members to 0.
    - void display() – a function that displays the information for this object. This function prints the text "Server computer:" and then calls the parent class display function. Then, for each array of IP addresses display a suitable heading and the phrase "List empty". In later assignments we will provide for connection to other computers and modify the display function.

Class **WAN**:

- data members in addition to those already in **node**:
    - an array of 4 IP addresses representing connections to server machines.
    - an array of 4 IP addresses representing connections to WAN machines.
    - two integers
- function members:
    - WAN(string n, IPAddress a) – an initializing constructor. This function calls the parent class constructor and then initializes the two integers to 0.
    - void display() – a function that displays the information for this object. This function prints the text "WAN computer:" and then calls the parent class display function. For each array of IP addresses display a suitable heading and the phrase "List empty". In later assignments we will provide for connection to other computers and modify the display function.

Part2: Expanding the switch in main.cpp

In main.cpp, declare a file-level (i.e., outside the main function) array of pointers to nodes called *network* of length MAX_MACHINES. At the beginning of the main function, initialize all elements of the array to be NULL before entering the main loop. After creating the above definted array, uncomment the lines at the end of the main function that deallocate the contents of the array.

Implement the CREATE_MACHINE case of the switch in your main program. The format of this command is

create_machine *type name IPAddress*

The acceptable values for *type* are "laptop", "server", and "wan". You will need a variable of type IPAddress to form a valid IPAddress object from the last token on the line.

- If the IPAddress object is not valid, do not create a new machine object but simply set your error_code variable to BAD_IP_ADDRESS and stop processing this command line.
- If the type specified by the second token is not one of the ones listed, do not create any

machine object but simply set your error code variable to UNKNOWN_MACHINE_TYPE and stop processing this command line.

- Search the network array for an element that is NULL. If none is found, set error code to NETWORK_FULL and stop processing this command line.
- Otherwise, create an object of the type specified and make the NULL position in network that you found point to the new object. You'll need to cast the item as a pointer to node. Then display the newly created object.

Implement the DESTROY_MACHINE case of your switch. The format of this command is:

destroy_machine *IPAddress*

Search the network array for a machine with the specified IP address. If none is found, set error code to NO_SUCH_MACHINE. Otherwise, destroy the object (i.e., use the C++ **delete** instruction) and set the element of the network array to NULL.

Modify the DATAGRAM case of the switch. Search the network array for a computer whose IP address is the source IP address specified in the command. If the source computer does not exist, set error code to NO_SUCH_MACHINE and stop processing this command line. Otherwise, use a variable of type pointer to datagram to create a new datagram object (using the C++ **new** instruction), and then have this new datagram object invoke makeDatagram with the information from this command line. Then have the source computer invoke initiateDatagram to store the newly created datagram.

Implement the SYSTEM_STATUS case of the main switch. (We will no longer have an integer as second token on this command line; that was strictly for testing your convert function.) System status should have each element of network that points to an object (i.e., is not NULL) display itself. Be sure to arrange the displays in an informative and easy to read format.

Test your program on the data in file p6input.txt. Notice that I have provided p6output.txt. I will be grading the output of your program on this assignment so pay attention to output.

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 code files + the text input file + 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 copied files, type 'make' to build your code.

3) Type **./Assignment** to run your code. Verify that the output is as you expect.

What to submit:

For this assignment, you'll be submitting both your machines.cpp and main.cpp but be SURE TO RENAME THEM as netid_main.cpp and netid_machines.cpp. For this assignment, the output must match EXACTLY to the provided output. Extra blank lines are acceptable, but otherwise, everything else must be exactly the same.