EECS 211 – Spring Quarter, 2010
Program 6

Due Tuesday, May 25th, 2010 at 11:59PM

In this assignment we will create an array of pointers to computer objects that will allow us to simulate a simple computer network. We will also define and implement three derived classes – three different kinds of computers, each with its own separate additional data members and, eventually, its own set of additional function members. This will be our simple way to model the real world, in which there actually are different kinds of computers with different capabilities. Finally, we will implement the three commands system_status, add_network_node, and delete_network_node.

Background:

Computer networks consist of a set of computers of different kinds that can communicate with each other. The specific way they communicate (i.e., the so-called physical and related layers) is not of concern to our project. We will simulate this in software. Our concern for the moment is that there are possibly many computers that can send files to each other. Moreover, there are different kinds of computers: personal computers (PCs, MACs, laptops), network servers, large file servers (characterized by huge disk drives) and a variety of other special purpose computers. In our project we will eventually simulate three different kinds of computers: PC, file server, and printer. Finally, computers can be turned on and off, the network cable can be plugged in or unplugged, etc. Therefore, the set of computers that are currently active in the network may be constantly changing. We will use an array to simulate a computer network.

Assignment:

(1) In definitions.h add a new constant, **MAX_NETWORK_NODES**, to your project and set it to 10. In system_utilities.cpp declare an array of pointers to **computer**. This will represent the network. The length of the array is **MAX_NETWORK_NODES**.

(2) Add three new classes, each derived from the **computer** class: PC, server, and printer. Each new class will have one additional data member over and above the data members of the base class **computer**:

- The PC class should have a character string of length up to 31 (requiring, of course, an array of <u>32</u> characters) to hold the name of the owner of this PC.
- The server class should have a character string of length up to 15 (requiring an array of 16 characters) to hold a string telling the location of this server.
- The printer class should have an integer variable telling how many printers are attached to this computer.

Each of these classes should have its own print function because each one has slightly different information to print. The declarations should be in machines.h, and the implementations in machines.cpp.

(3) Implement the commands: SYSTEM_STATUS, ADD_NETWORK_NODE, and DELETE_NETWORK_NODE. The formats for the first and last of these commands are:

```
    SYSTEM_STATUS:
        format:    system_status
        example:   system_status
    DELETE_NETWORK_NODE:
        format:    delete_network_node  name
        example:   delete_network_node  “larrys PC”
```

The format of the ADD_NETWORK_NODE command depends on what type of computer is being created. In particular, the second token on the line gives this information and determines how the last token on the line is interpreted. The formats are:

```
        add_network_node   PC       name   disk-size   owner
        add_network_node   server   name   disk-size   location
        add_network_node   printer  name   disk-size   number-of-printers
```

Examples are:

```
        add_network_node  PC   “larrys PC”   2000   “Larry Henschen”
        add_network_node   server   EECS1   5000   M357
        add_network_node   printer   “office printer”   3000   12
```

(4) In order to implement the delete node command you will need to be able to find the specified computer in the network, i.e., find the subscript j such that the jth element of your network array is a computer with name matching the one on the command line. To this end, first add a new function member to the **computer** class:

```
            int   isThisMyName(char *n);
```

This function returns 1 if the argument n matches the computer's name and returns 0 otherwise. Then add a new function to system_utilities (prototype in system_utilities.h, implementation in system_utilities.cpp) that accepts a string as argument and returns the subscript where that computer occurs in the network if it does occur and –1 if such a computer does not occur.

(5) In the command to enter a new computer into the network the disk size will be parsed by our token parser as a string. You will need to convert that string to an integer before actually creating the computer. Write a function that accepts a string as argument and returns the equivalent integer value. You may assume the string is correctly formed and do not need to do any error checking. YOU MAY NOT USE THE BUILT-IN C++ atoi FUNCTION.

Requirements and Specifications:

(1) The constructor for each of the derived classes should call the constructor for the base class to manage the common data members. Similarly, the print function in each class should call the print function in the base class to print the common information.

(2) Write three new functions – one to process each of the three commands we are implementing in this assignment. These functions - system_status, add_network_node, delete_network_node – as well as the functions for find a computer in the network and convert a string to integer, should be defined in the system_utilities cpp file. The prototypes for these should be placed in the system_utilities header file.

(3) The command system_status should cause every computer currently in the network to print itself, i.e., call its own **print** member function.

(4) The command add_network_node should first check if there is room in the network array to add another computer. If not, print an error message. Otherwise create a new object of the type indicated on the command line. Note, your program will need to look at the second token on the line to know what kind of object to create and how to interpret the last token on the line (owner name, location name, or integer telling number of printers). If the second token is not one of "PC", "server", "printer", do not create a new object but rather print an error message. Otherwise, create a new object of the correct type and add the address of that newly created object to the network array. Write a function in system_utilities to perform this operation. The arguments to the function should be the number of tokens on the command line and the list of parsed tokens and nothing else. In other words, the case in the switch in your main function should pass the token count and array of tokens directly into this new function. The switch case in the main function, then, consists of just a call to this function and a **break** statement.

(5) The command delete_network_node should search the network array for a computer with the indicated name. If one is found that computer should be removed from the network array and the object destroyed so that the memory it uses is given back to the system. If a computer with that name is not found, print a suitable message. Write a function in system_utilities to perform this operation. The arguments to the function should be the number of tokens on the command line and the list of parsed tokens and nothing else.

(6) Your project should process the command list in the file p6input.txt (http://www.eecs.northwestern.edu/~iraicu/teaching/EECS211/code/p6input.txt).

Comments, suggestions, and hints:

(1) Remember that the data members in the **computer** class should be **protected** and not **private** so that the derived classes can reference those members directly.

(2) As in the previous assignments you can do this one in easier steps.

1. Write and test the function that converts a string to an integer. You can write a simple main.cpp that reads lines that contain strings of digits, calls your function, and prints the result. Then throw away this main function.
2. Write and test the new derived classes. For this one you could use another throw-away main function, like we did for assignment 3. This main just has some variables that you use to create different kinds of computers and then call the public interface functions. You could also do these one at a time. Get the first one right, and the other two will be easy to add.
3. When the classes have been tested, bring back your main function with the token parsing and switch, add the network array and implement the add_network_node function and system_status case.
4. Finally, write the functions for finding and deleting network nodes.

(3) Converting a string of digits to a number can be done by processing the characters of the string left-to-right. An example will illustrate the process. Suppose the string is "953". Remember that the character representations for the digits '0'-'9' are 48-57 respectively, or in hexadecimal notation 30-39. Also remember that the string itself has a zero byte at the end. Thus, the above string would appear in the computer memory as a sequence of four bytes (in hexadecimal):

```
39  35  33  00
```

Set the variable that will collect the integer value to 0. Each time we process a digit we first translate the character to its integer equivalent; this simply means subtracting decimal 48. We then multiply the value collected so far by ten and add the new digit. In this case, we start with zero. After processing the first character we will have 9 (10*0 + 9). After processing the second digit we have 95 (10*9 + 5). After the third we have 953 (10*95 + 3). When we reach the zero byte we are done.

One way to translate the characters to their integer equivalents that is completely independent of the machine representation is to subtract '0', that is, the character zero rather than a decimal number. The compiler will insert a byte for the character '0', which, of course, will in fact be the integer representation of that character. C++ handles expressions in which **char** type variables and constants are used with arithmetic operators. Thus, assuming N is of type integer and D is of type **char**, the following assignment statement could be used in the conversion function:

```
N = N*10 + (D-'0');
```

(4) As our project contains more header files it can easily occur that one header file winds up being included in a given cpp file more than once. For example, you would likely include definitions.h in the system_utilities.cpp file. But now definitions.h may need to be included in disk_drive.h, which may in turn be included in computer.h, which may in turn be included in system_utilities.cpp. Thus, definitions.h got included twice, once

explicitly and once indirectly. C/C++ has a mechanism for avoiding such multiple inclusions – the #ifndef/#endif, to be used in conjunction with #define. The form of #ifdef is

```
                    #ifndef symbol
```

and the meaning is if the indicated symbol is <u>not</u> defined then process all the statements up to the matching #endif, otherwise skip all those statements. In header files the #ifndef is usually followed immediately by a #define symbol. The symbol itself is usually all uppercase and usually the name of the header file. For the definitions.h file you could have:

```
                    #ifndef          DEFINITIONS
                    #define          DEFINITIONS
                      …  the definitions
                    #endif
```

When processing a file, the compiler will not have seen the symbol DEFINITIONS the first time the header file is included, so #ifndef will be true and the definitions will be included. If that header file is included again, the #ifndef will be false because the compiler will have already seen the symbol DEFINITIONS.

(5) The test data will produce a large amount of output. This will be the case for all remaining assignments. To help you read the output it will be useful for you to be able to force your program to wait at strategic points, for example after each system_status command. Write a function called **wait** with no arguments. This function prompts the user to press ENTR to continue and waits until the user does so.