EECS 211 – Spring Quarter, 2010
Program 5

Due Tuesday, May 18th, 2010 at 11:59PM

In this assignment we will form the basic looping and branching mechanism in the main function that will be used for the remaining assignments in the project. We will also add the data structures and functions necessary to recognize the commands that our project will eventually be able to process. Our simulator will be driven by commands read from a text file. Our simulator will eventually be able to handle nine commands:

system_status
halt
add_network_node
delete_network_node
create_file
ls
delete_files
print_files
transfer_file

The forms of the remaining tokens for these commands and the exact nature of what they do is not important for this assignment. You will not be implementing any of these commands, just recognizing them.

Background:
Command-line interpreters normally treat the first token on a line as a command. The remaining tokens on the line are additional information to be used for that instance of that command. For a given command, different instances of that command may have different numbers of tokens. You are familiar with this concept in the UNIX system, where for example the g++ command can have varying numbers of additional tokens depending on which options you want for a particular compilation.

A typical command-line parser, then, works as follows. First parse the tokens on the line. Then use the first token to branch to code in the project that handles that command. Each individual command will have code that interprets or uses the remaining tokens on the line. In this assignment we will implement the first part of this process – parsing the tokens and using the first token to branch to different sections of our main function.

Assignment:

1. Add definitions for the following new constants to the definitions header file.

NUMBER_OF_COMMANDS	9
SYSTEM_STATUS	50
HALT	51
ADD_NETWORK_NODE	60
DELETE_NETWORK_NODE	61
CREATE_FILE	70

LS	71
DELETE_FILES	72
PRINT_FILES	73
TRANSFER_FILE	80
UNDEFINED_COMMAND	99

Note that UNDEFINED_COMMAND is not a command, so the number of commands that our system will process really is just 9. The constant NUMBER_OF_COMMANDS will be used to declare an array to hold the information needed about our commands. The remaining constants will be used as **case** labels for a **switch** statement in the main function.

2. In system_utilities.cpp define a new class **commandElement** as follows:
- data members (These may be public, or else you can also include appropriate access functions to accomplish the requirements of the function **getCommandNumber** described below.)
    - a pointer to **char** – this will hold the string representation of the command, for example "ls" or "add_network_node".
    - an **int** – this will hold the integer representation of the that token, for example CREATE_FILE.
- function member
    - **commandElement(char *, int)** – The constructor copies the two arguments to the two data members. Of course, you will have to malloc space to hold the actual command string.

Note that this class will be used only inside system_utilities.cpp. It is therefore declared in system_utilities.cpp, not system_utilities.h, so that it can't be included into other cpp files in the project.

3. Declare a file-level variable in system_utilities.cpp of type array of pointers to **commandElement** of length NUMBER_OF_COMMANDS. (NOTE: The constant UNDEFINED_COMMAND does not correspond to a command. It is used as a return value on the function **getCommandNumber**, described below, when its argument s points to a token which is not a command.)

4. Write the following two new functions in system_utilities.cpp, with corresponding prototypes in system_utilities.h.

void fillCommandList()
This function fills the array of **commandElement** object pointers with the following list:

"system_status"	SYSTEM_STATUS
"halt"	HALT
"add_network_node"	ADD_NETWORK_NODE
"delete_network_node"	DELETE_NETWORK_NODE
"create_file"	CREATE_FILE
"ls"	LS

"delete_files"	DELETE_FILES
"print_files"	PRINT_FILES
"transfer_file"	TRANSFER_FILES

This function will be called once from the main function before it enters the loop to read the input file.

int getCommandNumber(char *s)
This function searches the array for an element whose string data member matches the string pointed to by s. If a matching element is found, return the corresponding integer data member. If no match is found return UNDEFINED_COMMAND.

5. Write a new main function that first attempts to open the file p5input.txt (http://www.eecs.northwestern.edu/~iraicu/teaching/EECS211/code/p5input.txt). If the file is not opened, main should quit. Otherwise, main calls **fillCommandList**. The main function then continues to read lines until the line beginning with the token "halt" is encountered. In the loop your program should parse the line into tokens, search for the first token in the list of commands, and branch to an appropriate **case** in a **switch**. Each case of the switch should print a message saying what command was recognized and print a list of all the tokens on that command line, one token per line. For example, if the input line was

```
add_network_node    PC    DELLL104    8096   "Larry's PC"
```

your program should generate output like

```
Recognized command to add a new node to the network.
The tokens were:
        add_network_node
        PC
        DELL104
        8096
        Larry's PC
```

Requirements and Specifications:

1. The switch statement in your main function should use the defined constants as case labels. So, your switch should look like:

```
switch( … ) {
        case   SYSTEM_STATUS:  ….
                                break;
        case              HALT:
                                break;
        …
```

}

Comments, suggestions, and hints:

1. Adding the defined constants to your header file is easy and should be done first.

2. You could do the switch in the main program next. You wouldn't have the command array or the function that finds the command number, but you could still add and test the switch in the following way. In place of code that reads and parses lines of text and the code at the end that frees that "malloc"-ed strings write a simple prompt for an integer input. Then use that integer as the switch variable on your case statement. This would allow you to test each of the 10 cases (the nine actual commands and the UNDEFINED_COMMAND case) to see if the switch was working and you liked the first line of output. When you are satisfied with that much, you can throw away the prompt and integer input, add the code that reads from the file and parses the tokens, and continue with the next step.

3. The next step is to declare and initialize the array. I would use the debugger to step through the code and then look in the watch or data window to examine the values in the array elements to see if they are what you want.

4. Finally, implement the function that finds the command number from the input and add a call to that function in the main program before the switch.

Test data for p5input.txt (http://www.eecs.northwestern.edu/~iraicu/teaching/EECS211/code/p5input.txt**):**

```
add_network_node PC pc1 4096 Larry
add_network_node printer pr1 2048 5
system_status
create_file pc1 f1 20
create_file pc1 f2 70
this is not a command
create_file pc1 f3 70
delete_files pc1 f1 f2
delete_network_node pr1
ls pc1
transfer_file pc1 pr1 f1
delete_files s1 "hello mom" myproject
this is also not a command
transfer_file pc1 pr1 f3
ls pr1
print_files pr1
transfer_file pc1 pc2 f3
transfer_file pc1 pr1 f4
system_status
```

halt