Larry Bush
Student ID: 660 220 742
Email: bushl2@rpi.edu or Lawrence_Bush@dps.state.ny.us

April 26, 2002

Final Project
Operating Systems
Grep for Windows

Table of Contents:

Project Description

This program is something like the grep command in UNIX. It searches through the files in a given directory (including sub-directories) for a word and locates each occurrence of the word. It can also replace each occurrence of that word with another word at the user's request.

This program is multi-threaded and preemptable (you can stop a search that is in progress).
The program also maintains data integrity at all times by storing a temporary copy of the file that it is doing a search and replace on. It deletes this file after the search file is safely edited and saved.

This is an Individual Project (No Partners).

There are no known problems with the program. It does everything in the Spec.

Compiling

This program workspace is set up to compile in Visual C++ in Multi-threaded Debug mode.
It is also set up to process the first Test Sequence (see below).

Running

This program is oriented towards UNIX users who wish windows had grep with replacement. Therefore, the user interface accepts commands in a similar way to the UNIX grep.

Run from the DOS Prompt:

To run the program, the user will type in the command grep (with arguments) at the dos prompt. The program must be in the current directory.

The user will input the command line in the following format:

```
grep      word_to_find      [replacement_word]      directory
```

The replacement word is optional.

For example, if I want to search for the word school in all the files in my letter folder, I type:

```
grep school C:\Windows\Desktop\Letters
```

Then, if I want to change the word school to college in all my letters that I have written, I could type:

```
grep school college C:\Windows\Desktop\Letters
```

The word to find and the replacement word should each be a single word (no spaces). If you wish to search for or replace with a phrase (with spaces), then it must be enclosed in quotes.

You will be allowed to review your request before proceeding.

You can also stop a search in progress by typing 'y' and return.

For each match found, the program will output:
- File name
- The line the word is found on.

Run from Visual C++:

This program can also be done run through Visual C++ by opening the Project-Settings menu, clicking on the Debug tab, and putting the arguments into the "Program Arguments" box. Note that you leave out the grep command and just input the arguments.

Sample Output

The zip file I submitted contains a hierarchy of subfolders with files in them. Four of the files in the folder "layer2" contain the word "searchWord" in them. The workspace is currently set up to run this :

```
searchWord .\files_to_find\layer2
```

If you run it, you will get the following results:

```
GREP - SEARCH ONLY
------------------

Search directory:
-----------------
.\files_to_find\layer2\

     Word To Find      : searchWord

Note: You can stop a search in progress by typing 'y' and return.
Continue?  y/n  > y
Type y and return to stop search. >

MATCH
File: searchfileBottomLevel.h
Line: 1111111111111searchWord111111111

MATCH
File: searchfile3.h
Line: 1111111111111searchWord111111111

MATCH
File: searchfile2.h
Line: 1111111111111searchWord111111111

MATCH
File: searchfile.h
Line: 1111111111111searchWord111111111

Search Completed.  Quit?  y/n  >
```

Program Test Sequence

The following are 4 more tests that allow you to test the REPLACE function, and the PREEMPT function.

Test 1 : This tests the REPLACE function of the program. The following arguments direct the program to replace the word "searchWord" with the word "OSisFunFunFun".

Arguments: **`searchWord OSisFunFunFun .\files_to_find\layer2`**

If you run grep with these arguments, you will get the following results (it will find 4 matches) :

```
GREP - SEARCH and REPLACE
-------------------------

Search directory:
-----------------
.\files_to_find\layer2\

     Word To Find      : searchWord
     Replacement Word  : OSisFunFunFun

Note: You can stop a search in progress by typing 'y' and return.
Continue?  y/n  > y
Type y and return to stop search. >

MATCH
File: searchfileBottomLevel.h
Old Line: 1111111111111searchWord111111111
New Line: 1111111111111OSisFunFunFun111111111

MATCH
File: searchfile3.h
Old Line: 1111111111111searchWord111111111
New Line: 1111111111111OSisFunFunFun111111111

MATCH
File: searchfile2.h
Old Line: 1111111111111searchWord111111111
New Line: 1111111111111OSisFunFunFun111111111

MATCH
File: searchfile.h
Old Line: 1111111111111searchWord111111111
New Line: 1111111111111OSisFunFunFun111111111

Search Completed.  Quit?  y/n  >
```

Test 2 : This tests the REPLACE function again, but references the file from the root directory. For this to work on your computer, you must input the absolute address specific to your directory hierarchy.

The following arguments direct the program to replace the word "OSisFunFunFun" with the word "searchWord" .

Arguments: **`OSisFunFunFun searchWord  C:\WINDOWS\Desktop\Grep13\files_to_find`**

If you run grep with these arguments, you will get the following results (it will find 4 matches) :

```
GREP - SEARCH and REPLACE
-------------------------

Search directory:
-----------------
C:\WINDOWS\Desktop\Grep12\files_to_find\

     Word To Find      : OSisFunFunFun
     Replacement Word  : searchWord

Note: You can stop a search in progress by typing 'y' and return.
Continue?  y/n  > y
Type y and return to stop search. >

MATCH
File: searchfileBottomLevel.h
Old Line: 1111111111111OSisFunFunFun111111111
New Line: 1111111111111searchWord111111111

MATCH
File: searchfile3.h
Old Line: 1111111111111OSisFunFunFun111111111
New Line: 1111111111111searchWord111111111

MATCH
File: searchfile2.h
Old Line: 1111111111111OSisFunFunFun111111111
New Line: 1111111111111searchWord111111111

MATCH
File: searchfile.h
Old Line: 1111111111111OSisFunFunFun111111111
New Line: 1111111111111searchWord111111111

Search Completed.  Quit?  y/n  >
```

Test 3 : This tests the preempt function.

The following arguments direct the program to search for the word “pookator” with the word “searchWord” .

Arguments: **`pookator .\files_to_find`**

The search starts one layer up from the previous search and as a result also searches the directory “deepsearchdir”. This is a deep directory with many files. The search finds a match to :pookator” toward the end of the search. Therefore, this can be used to test the abort command.

To do this:

- Run grep with this command and you will see that it finds the match.
- Then run it again, and abort. (y and return)

You will then see that the search stops the search before finding the match (if you do it quick enough).

Results without abort:

```
GREP - SEARCH ONLY
------------------

Search directory:
-----------------
.\files_to_find\

     Word To Find      : pookator

Note: You can stop a search in progress by typing 'y' and return.
Continue?  y/n  > y
Type y and return to stop search. >

MATCH
File: Copy of Copy of Copy of longFile.h
Line: template <class Iterator, clapookatorss T>

Search Completed.  Quit?  y/n  >
```

Results with abort:

```
GREP - SEARCH ONLY
------------------

Search directory:
-----------------
.\files_to_find\

     Word To Find      : pookator

Note: You can stop a search in progress by typing 'y' and return.
Continue?  y/n  > y
Type y and return to stop search. > y
Quitting...
Press any key to continue
```

Test 4 : This tests grep from the DOS Prompt.

Enter the following arguments into the DOS Prompt Window. For this to work, you must be in the Debug directory of the Grep13 folder.

Arguments: **`grep searchWord ..\files_to_find\layer2`**

The search moves one layer up to the parent folder, then down into the "layer2" directory.
This is the same search as the very first example, except that it is done from the DOS prompt.

Results:

```
C:\WINDOWS\Desktop\Grep13\Debug>grep searchWord ..\files_to_find\layer2

        GREP - SEARCH ONLY
        ------------------

        Search directory:
        -----------------
        ..\files_to_find\layer2\

             Word To Find      : searchWord

        Note: You can stop a search in progress by typing 'y' and return.
        Continue?  y/n  > y
        Type y and return to stop search. >

        MATCH
        File: searchfileBottomLevel.h
        Line: 1111111111111searchWord111111111

        MATCH
        File: searchfile3.h
        Line: 1111111111111searchWord111111111

        MATCH
        File: searchfile2.h
        Line: 1111111111111searchWord111111111

        MATCH
        File: searchfile.h
        Line: 1111111111111searchWord111111111

        Search Completed.  Quit?  y/n  > y
        Quitting...

        C:\WINDOWS\Desktop\Grep13\Debug>
```

Program File Description

File	Purpose
grep.cpp	Initializes the chat server. Calls functions to accept new client connections and handle them.
traverse.h	Declares the traverse class, which traverses all the subdirectories of a given subdirectory. It also uses the buffer class to process each file that it finds. It performs these tasks on a separate thread from the main program. The traverse class manages its own thread. In other words, the thread is called by a member function of the class. This thread is preemptable. In other words the user may stop the grep process in mid stream. The idea behind this is that the user may have made a mistake and invoked a process that will take a very long time or possibly have unintended (bad) consequences. The traverse class also handles this functionality.
traverse.cpp	Implementation of traverse member functions.
buffer.h	Declares and implements the buffer class. The buffer class is called by the traverse function and processes a given file. Processing the file includes opening the file, reading the file into memory (using the line class to store each line), storing a temporary copy of the file to disk, searching the file for a user specified search word (line by line using the line class), replacing the word with the user specified replacement word (if provided), saving the revised file back to disk, closing the file and deleting the temporary file.
line.h	Declares and implements the line class. The line class is used to store and process a line of the file buffer. Processing a line includes storing the line, providing access functions to the line, provide functions for searching the line for a user specified search word, and replacing the word with the user specified replacement word (if provided).

Program Design

Class Organization

This program uses 3 classes: traverse, buffer, and line.

Traverse	The traverse class contains functions that traverse all the subdirectories of a given directory and handle the thread operations.
Buffer	The buffer and line classes together store and process the files that the program searches.
Line	The line class is used by the buffer class to store and manipulate one line of a file.

Program Flow

The flow of the program is as follows:

The Main() function gets user input through the input arguments. It reports an error if the user inputs an incorrect number of arguments. It then creates a traverse object and sets the user input variables of the traverse class. It then calls the multithreaded_traverse() function of the traverse class to start the operation.

The multithreaded_traverse() function starts the search thread and calls the thread function.

Directory Searching

The thread function sets some intermediate variables and calls the DirSearchHelper() function. The DirSearchHelper() and the DirectoryOrFile() function work together to traverse the directories and search the files. (These are both member functions of the traverse class.) A directory path is passed to the DirSearchHelper(). That function then iterates through each file/directory within that directory. For each file/directory (except . and ..) that it finds in the directory, the function calls the DirectoryOrFile() function. This function checks to see if the item is a directory or file. If it is a directory, it recursively calls DirSearchHelper() so that it can iterate through that sub-directory. If the item is a file, it creates a buffer object and calls buffer::grep_search_only() or buffer::grep_with_replacement(). These buffer member functions open the file and read it into memory using the line class to store each line and the line::get_lines_from_file(file) member function. The buffer then stores a temporary copy of the file to disk to protect data integrity.

Word Replacing

Once the file has been read in and a temp file has been saved, the program will search (or search and replace) each line in the file for a user specified search. The user specifies if he wants the search option or the search and replace option via the user input program arguments. If the user includes a replacement word in his arguments, then the program will invoke the search and replace option. If the user does not specify a replacement word in his program arguments, then the program will only invoke the search option. This is accomplished by the main() program by passing a special token to the traverse class.

The function - line::match_first() performs the search operation. The line::match_first() function searches for a word in the line and prints the file name and the line to the screen if there is a match. This is an O(M*N) algorithm where M is the word length and N is the line length. However, the best case is done in M time and the average case is close to N time.

The function line::replace_matches_on_this_line() performs the search and replace operation. The line::replace_matches_on_this_line() function searches for a word in the line and prints any matches to the screen. This is also an O(M*N) algorithm where M is the word length and N is the line length. The best case is N and the average case is close to N time.

If a replace was done, the file will be re-written and saved.

The buffer class destructor deletes the temporary file when these operations for that file are completed.

The process then returns to the recursive traverse. Once all sub-directories have been traversed and all directories are iterated through and all files are searched, the program returns to the thread function and subsequently back to main().

Preemption

This program allows the user to stop the search while it is in progress. The preemption operation is accomplished as follows. The program starts a user space thread. There is a global flag variable (stop) that is initially set to 0. The calling thread asks the user if he would like to stop the currently running search. If the user types y (for yes) and return, then the main calling thread will set the stop variable to 1.

The search thread periodically checks this variable. If the stop variable is still 0, then it continues as usual. If the stop variable is 1, then it returns immediately from the search thread.

It does this in between each file search. It is done in between files to preserve data integrity. It also ensures that we do not leave behind a temporary file.

Resource Management

- The ~destructors for the traverse, buffer, and line classes delete all directly allocated memory.

- The buffer ~destructor also deletes the temporary file from the hard disk.

- Most of the member data in this program are declared as objects rather than pointers and therefore are automatically deleted when they go out of scope.

- My traverse object closes all handles when it no longer needs them.

- My buffer object deletes all of its line objects before closing.

Error Checking

I check for errors on the following System Calls:

WaitForSingleObject
CloseHandle
_beginthreadex
DeleteFile
FindFirstFile
FindNextFile

Attachment 1: Project Code

Attachment 2: Project Spec.