CSC Program 3
Due April 13 at Midnight

Program Description:

A company would like a detailed xml file produced of a directory. They are interested in files that match a particular pattern. When a file is found matching the pattern, you may have to output the creation date, the modification date, or both. This xml file can then be opened up in a web browser to view the results.

Program Execution:

Your program will be started from the command line using the following syntax.

C:\prog3.exe [option] directory pattern

Option:

-C show creation date
-M show modification date
If no option is given, do not output any dates to the xml file.

Directory:

This is the starting point for the xml file creation.

Pattern:

Used to filter which files will be outputted to the XML file.

Examples
Prog3.exe c:\Source *.cpp
Prog3.exe SourceDirectory p*.h
Prog3.exe –M –C Source *.h
Prog3.exe –C –M Source *.cpp
Prog3.exe –M \project *.cpp
Prog3.exe –C startups *.*

You must check that the directory exists and that a valid option has been given.

XML File:

Your output will be stored in a file named “dir.xml” that will be located in the directory from which the program was started. Open the file before changing to a directory. After opening the file for output, the very first line in the will read:

<?xml version= "1.0"?>

The double quotes are critical.

Then you will make a function call to create the recursive xml file. Each folder will have an xml tag that displays the full path to the folder. Before closing the folder tag, you will output all its subfolders, and the files that match the pattern along with any dates. Each Subfolder will then have a function call to create is xml structure. Files will come after the folder listing in the xml file.

XML Folder Tags

<folder name="full path to directory"> ← Starts the folder tag

Listing of this folders contents

</ folder> ← ends the folder tag

The folder tag will have other folder and file tags within it. Each folder tag could then have more folder and file tags within it.

Xml File Tags

No dates

<file name="filename" /> ← < /> means nothing inside the tag so a separate Closing tag is not needed.

-C flag specified

<file name="filename" DateCreated="Mon May 31 16:04:29 2010" />

-M flag specified

<file name="filename" DateModified="Mon Mar 19 16:04:29 2012" />

-C -M flag specified

<file name="filename" DateCreated="Mon May 31 16:04:29 2010" DateModified="Mon Mar 19 16:04:29 2012" />

Example: if you had a folder csc250 folder that contained your 4 programming assignments. With c++ & h files inside and program documents in the base directory.

The program might be started like this:

```
Prog3.exe c:\classdata\csc250  *.*
```

Output dir.xml produced

```
<?xml version= "1.0"?>
<folder name="C:\classdata\csc250\">
    <folder name="C:\classdata\csc250\program 1\">
        <file name="functions.cpp" />
        <file name="functions.h" />
        <file name="prog1.cpp" />
    </folder>
    <folder name="C:\classdata\csc250\program 2\">
        <file name="mylist.cpp" />
        <file name="mylist.h" />
        <file name="prog2a.cpp" />
        <file name="prog2b.cpp" />
    </folder>
    <folder name="C:\classdata\csc250\program 3\">
        <file name="prog3.cpp" />
    </folder>
    <folder name="C:\classdata\csc250\program 4\">
        <folder name="C:\classdata\csc250\program 4\version 1.1\">
            <file name="prog4.cpp" />
            <file name="queue.cpp" />
            <file name="queue.h" />
        </folder>
        <folder name="C:\classdata\csc250\program 4\version 1.2\">
            <file name="prog4.cpp" />
            <file name="queue.cpp" />
            <file name="queue.h" />
        </folder>
        <folder name="C:\classdata\csc250\program 4\version 1.3\">
            <file name="prog4.cpp" />
            <file name="queue.cpp" />
            <file name="queue.h" />
        </folder>
        <file name="prog4.cpp" />
        <file name="queue.cpp" />
        <file name="queue.h" />
    </folder>
    <file name="prog1.docx" />
    <file name="prog2.docx" />
    <file name="prog3.docx" />
    <file name="prog4.docx" />
</folder>
```

Or like this: give me all files that start with p and have a cpp extension

Prog3.exe –M c:\classdata\csc250 p*.cpp

Dir.xml produced

```
<?xml version= "1.0"?>
<folder name="c:\classdata\csc250\">
    <folder name="c:\classdata\csc250\program 1\">
        <file name="prog1.cpp" DateModified="Mon Mar 19 15:46:24 2012" />
    </folder>
    <folder name="c:\classdata\csc250\program 2\">
        <file name="prog2a.cpp" DateModified="Mon Mar 19 15:46:43 2012" />
        <file name="prog2b.cpp" DateModified="Mon Mar 19 15:46:47 2012" />
    </folder>
    <folder name="c:\classdata\csc250\program 3\">
        <file name="prog3.cpp" DateModified="Mon Mar 19 15:47:11 2012" />
    </folder>
    <folder name="c:\classdata\csc250\program 4\">
        <folder name="c:\classdata\csc250\program 4\version 1.1\">
            <file name="prog4.cpp" DateModified="Mon Mar 19 15:47:19 2012" />
        </folder>
        <folder name="c:\classdata\csc250\program 4\version 1.2\">
            <file name="prog4.cpp" DateModified="Mon Mar 19 15:47:19 2012" />
        </folder>
        <folder name="c:\classdata\csc250\program 4\version 1.3\">
            <file name="prog4.cpp" DateModified="Mon Mar 19 15:47:19 2012" />
        </folder>
        <file name="prog4.cpp" DateModified="Mon Mar 19 15:47:19 2012" />
    </folder>
</folder>
```

File system traversal:

You will need to read the “FileSystem.docs” to understand how to recursively traverse a directory structure. All the functions Discussed in that document have been written for you. Just include the appropriate header files and use the functions.

The 4 fields that we will be using in the _finddata_t are:

- name – a c style string that contains the file / folder of files name
- attrib – number that contains the attributes of the folder or file. You must look at the appropriate bit to find out the information. If the fifth significant bit is set it is a folder, otherwise it is a file.
- time_write – the last date and time that the file / folder was modified.
- time_create – the date and time that file / folder was created.

Note: the time fields are a floating point number. You must do some research to find out how to convert them to human readable dates and times.

It is advisable to have a function that lists the files that match the pattern. Then your recursive function can call it right after it handles the folders.

When detecting your list of folders from the directory listing, there are two very important folders that you should not handle. The "." and the "..", these two names will cause your program to do infinite recursion. The single dot is a reference to the directory itself and the double dot is a reference to the parent directory. If the name (say a) is not one of these directories, change into it and then list its contents. When done, back out from (a). The hardest part of this routine is keeping your current working directory in the same spot as where you are reading from.

Algorithm:

Check Argument count
Check valid option if given
Open xml file.
Change to directory
Make call to recursive function
    Output folder tags
    Handle folder within
    Output the file listings matching the pattern.
Exit program.

TimeLine:

March 26, have the command line arguments parsed.
March 30, have error checked the dir.xml file opening and that the specified directory exists.
April 2, output a valid XML file without doing a recursive call for the inner folders. This includes the files that match the specified pattern.
April 7, be able to recursively traverse the directory structure outputting the xml tags.
April 10, have tested and debugged your program.
April 13, Program is due by 11:59pm.