Contents - Recursion lab

1 Setup **2**
1.1 Turn-in instructions 2
1.2 Setup 2
1.2.1 Running the program 3
1.3 Program design 4
1.3.1 Files and Folders 5
1.3.2 FileSystemManager 6
1.3.3 Logger 8
1.3.4 DisplayFolder recursive functionality 9
1.4 Recursive Find functions 18
1.4.1 Folder* Recursive_FindFolder(Folder* ptrFolder, string name) 19
1.4.2 File* Recursive_FindFile(Folder* ptrFolder, string name) 22

Part 1

Setup

1.1 Turn-in instructions

- Each lab in this course should have a corresponding folder in your course repository.
- Copy the URL to your repository in the folder for this assignment and paste that URL as the submission in Canvas.

1.2 Setup

There is a starter zip file provided which includes a bunch of source files for a program that you will modify. The zip file includes Code::Blocks and Visual Studio project files, and source files for the program.

See the next page for the list of all starter files.

Starter files:

```
Recursion_Lab/
├── main.cpp
├── File.hpp
├── File.cpp
├── FileSystemManager.hpp
├── FileSystemManager.cpp
├── FileSystemThing.hpp
├── FileSystemThing.cpp
├── Folder.hpp
├── Folder.cpp
├── tester/
│   └── utilities/
│       ├── Menu.hpp
│       ├── StringUtil.hpp
│       └── Logger.hpp
├── Project_CodeBlocks/
│   └── Code Blocks files
└── Project_VisualStudio/
    └── Visual Studio files
```

1.2.1 Running the program

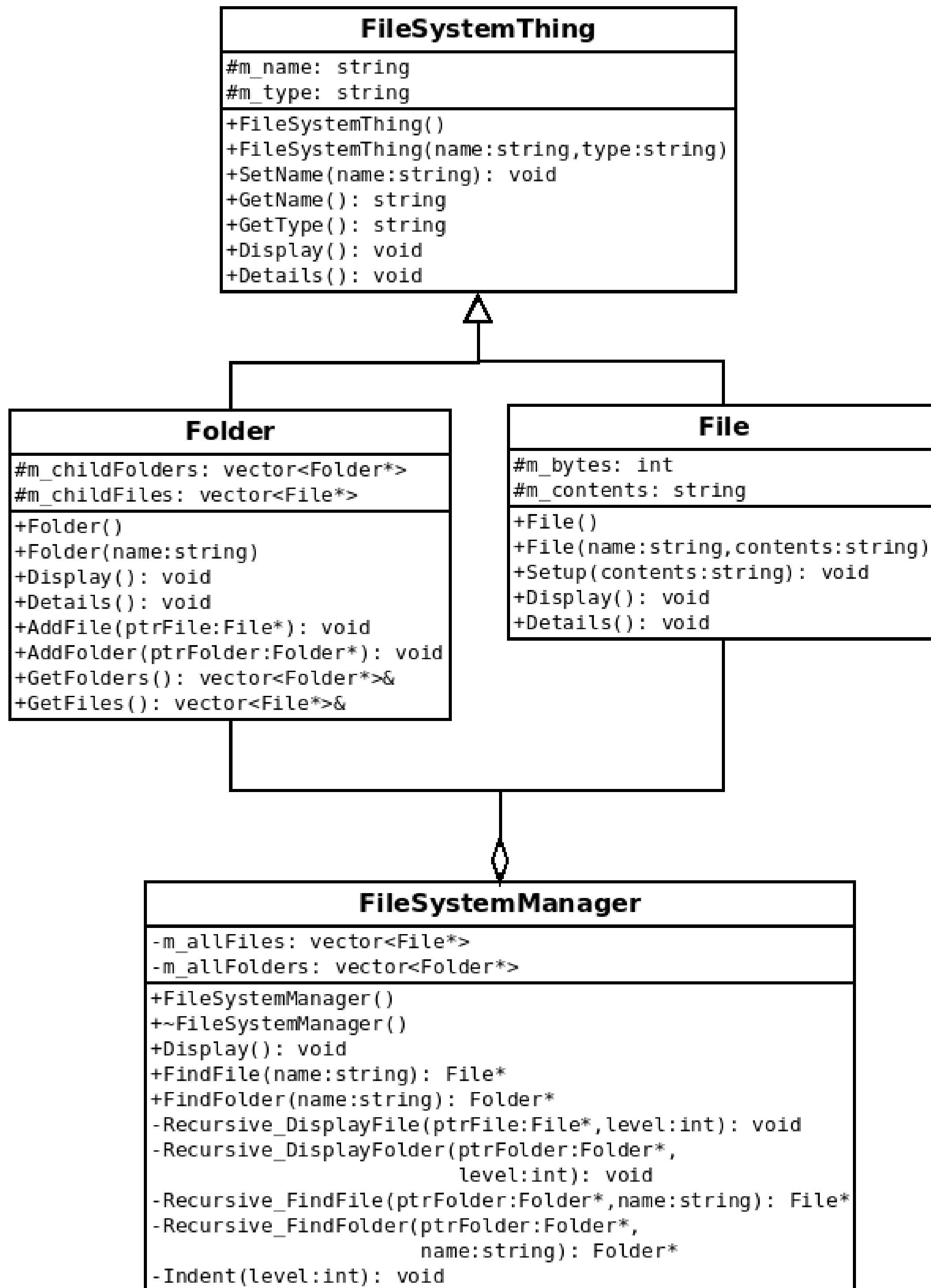
Most of this program is already implemented. When you run it, you will have a menu of options. The program already has everything implemented *except* the two search functions, which you will implement.

```
-----------------------------------------------------
 | MAIN MENU |
 -------------


 1. Search for file
 2. Search for folder
 3. View filesystem
 4. Quit

 >>
```

1.3 Program design



1.3.1 Files and Folders

The **FileSystemThing** class is the parent to the **File** and **Folder** class. It stores common variables, such as the file/folder name.

```
FileSystemThing
---
#m_name: string
#m_type: string
---
+FileSystemThing()
+FileSystemThing(name:string,type:string)
+SetName(name:string): void
+GetName(): string
+GetType(): string
+Display(): void
+Details(): void
```

The **File** class represents a file on the harddrive, which contains some form of contents (in this case, a string) and a size (in bytes).

```
File
---
#m_bytes: int
#m_contents: string
---
+File()
+File(name:string,contents:string)
+Setup(contents:string): void
+Display(): void
+Details(): void
```

The **Folder** class represents a folder, and may contain files and folders within it.

For this class, it stores a list of File pointers and Folder pointers.

```
Folder
---
#m_childFolders: vector<Folder*>
#m_childFiles: vector<File*>
---
+Folder()
+Folder(name:string)
+Display(): void
+Details(): void
+AddFile(ptrFile:File*): void
+AddFolder(ptrFolder:Folder*): void
+GetFolders(): vector<Folder*>&
+GetFiles(): vector<File*>&
```

1.3.2 FileSystemManager

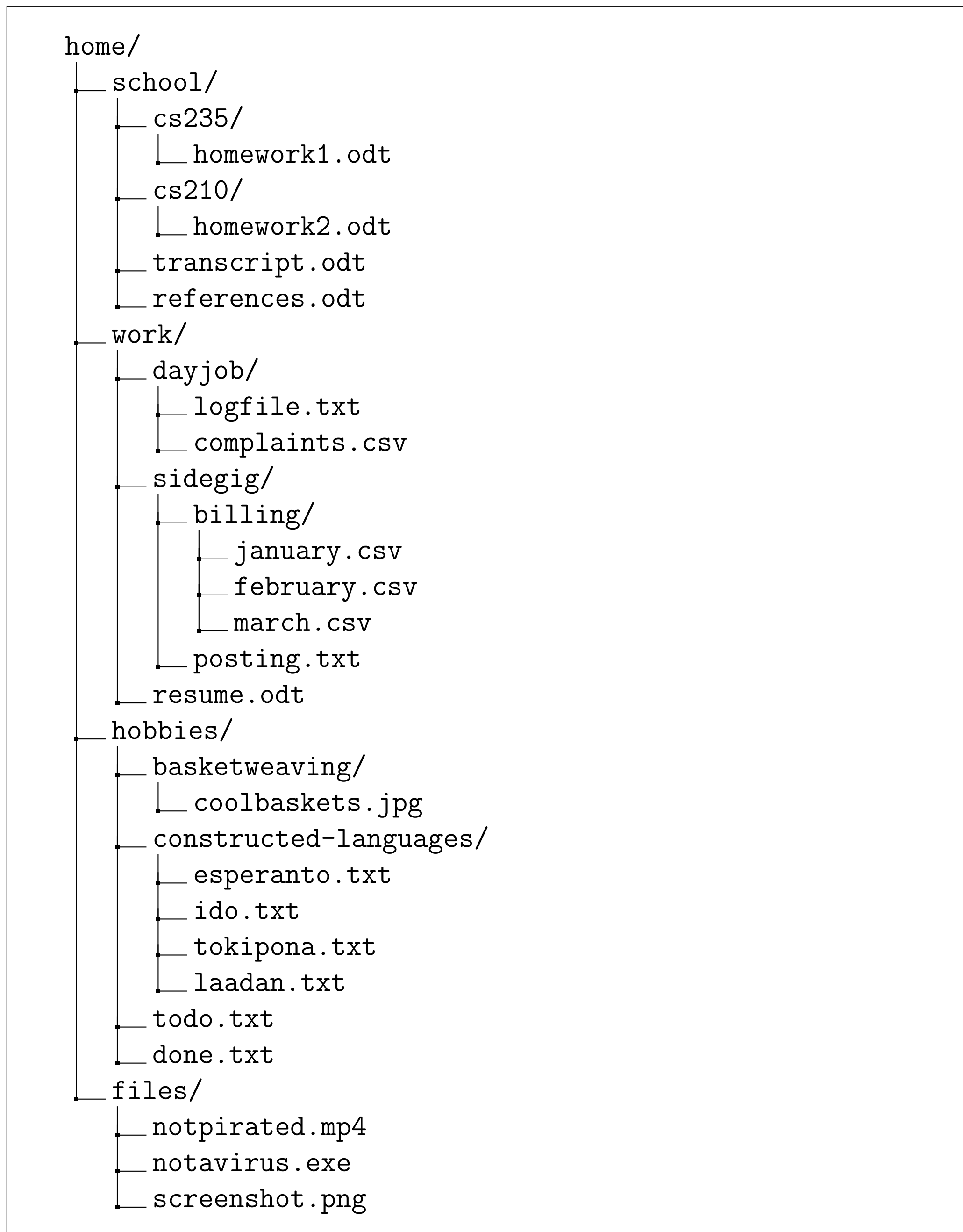
```
FileSystemManager
-----------------------------------------
-m_allFiles: vector<File*>
-m_allFolders: vector<Folder*>
-----------------------------------------
+FileSystemManager()
+~FileSystemManager()
+Display(): void
+FindFile(name:string): File*
+FindFolder(name:string): Folder*
-DisplayFile(ptrFile:File*,level:int): void
-Recursive_DisplayFolder(ptrFolder:Folder*,
                         level:int): void
-Recursive_FindFile(ptrFolder:Folder*,name:string): File*
-Recursive_FindFolder(ptrFolder:Folder*,
                      name:string): Folder*
-Indent(level:int): void
```

The FileSystemManager contains all the files and folders in the program, which are also set up within the FileSystem Manager.

Everything is already implemented, except the **Recursive_FindFile** function and the **Recursive_FindFolder** function. These functions will begin at the **home** directory (which is `m_allFolders[0]`) and continue searching each subfolder until the folder or file being searched for is found. If it cannot find the item, it will return `nullptr` instead.

You will also be able to use the **Recursive_DisplayFile** and **Recursive_DisplayFolder** functions for reference as well. The main difference is that these two functions have a `void` return type.

The mock filesystem is structured like this:



1.3.3 Logger

The Logger utility is also included in this program, which you can use to investigate how the program executed some functionality.

The **log.html** file is generated in your **project folder:** either the `Project_VisualStudio/RecursionLab/` directory or the `Project_CodeBlocks/` directory.

The log file looks like this:

13:30:06	FileSystemManager::FileSystemManager	Folder 3: cs210
13:30:06	FileSystemManager::FileSystemManager	Folder 4: work
13:30:06	FileSystemManager::FileSystemManager	Folder 5: dayjob
13:30:06	FileSystemManager::FileSystemManager	Folder 6: sidegig
13:30:06	FileSystemManager::FileSystemManager	Folder 7: billing
13:30:06	FileSystemManager::FileSystemManager	Folder 8: hobbies
13:30:06	FileSystemManager::FileSystemManager	Folder 9: basketweaving
13:30:06	FileSystemManager::FileSystemManager	Folder 10: constructed-languages
13:30:06	FileSystemManager::FileSystemManager	Folder 11: files
13:30:08	FileSystemManager::FindFile	Find file: todo.txt
13:30:08	FileSystemManager::Recursive_FindFile	Searching folder: home
13:30:08	FileSystemManager::Recursive_FindFile	Searching folder: school
13:30:08	FileSystemManager::Recursive_FindFile	Searching folder: cs235
13:30:08	FileSystemManager::Recursive_FindFile	Searching folder: cs210

It logs the time, the location of where the message is coming from (what function was called), and a note about what is happening.

A log entry on the code-side looks like this:

```
1 // Function header
2 void Logger::Out( const string& message,
3     const string& location )-
4
5 // Function call
6 Logger::Out( "Display file: " + ptrFile->GetName(),
7     "FileSystemManager::Recursive_DisplayFile" );
```

1.3.4 DisplayFolder recursive functionality

Let's step through how the DisplayFolder functionality works, which will help you think of how to implement the **FindFile/FindFolder** functionality.

We have two functions related to displaying files: a **public Display** function, and a **private Recursive_DisplayFolder** function.

void Display(): The public Display() function is the entry point and kicks off the recursive sequence. We always want to begin by displaying the filesystem from the root (home) folder, so we don't give the user a choice to start from a different folder (though we could).

```
void FileSystemManager::Display()
{
    Logger::Out( "Display filesystem", "
   FileSystemManager::Display" );
    // Start at root
    Recursive_DisplayFolder( m_allFolders[0], 0 );
}
```

void Recursive_DisplayFolder(Folder* ptrFolder, int level): This is the recursive function. It is given a pointer to *some* folder, and it will display itself before displaying all of its child files and folders.

The `level` variable is just for setting a level of indentation when displaying the filesystem, so that it appears tree-like.

Displaying the Files: A simple loop iterates through all the File pointers that this current `ptrFolder` Folder contains, using a **range-based for loop**:

```
for ( auto& ptr : ptrFolder->GetFiles() )
{
    DisplayFile( ptr, level+1 );
}
```

We could have also written the code this way:

```
vector<File*> files = ptrFolder->GetFiles();
for ( unsigned int i = 0; i < files.size(); i++ )
{
    DisplayFile( files[i], level+1 );
}
```

Displaying the Folders: A file has no additional depth to it; the Folder's child files are all at the same "depth". However, a Folder can also contain subfolders within it.

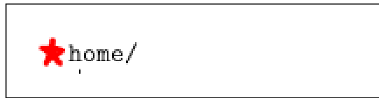
For each subfolder we have, we will call this same function (recursing into it) in order to perform the same operations from the next depth-level: Display self, display subfolders, display files.

```
for ( auto& ptr : ptrFolder->GetFolders() )
{
    // Recurse into the same function
    Recursive_DisplayFolder( ptr, level+1 );
}
```

Stepping thru the recursion

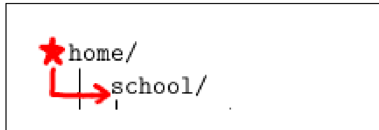
Let's step through how the filesystem is displayed, step-by-step, so we can see how recursion works. You may want to use the diagram of the mock filesystem from page 7 for reference as we do this.

1.



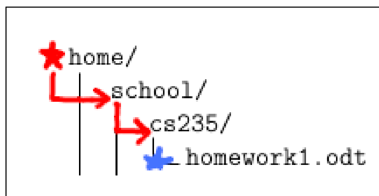
Recursive_DisplayFolder("home")
Write "home" to the screen.
Recurse into first subfolder...

2.



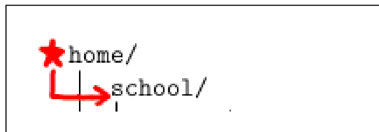
Recursive_DisplayFolder("school")
Write "school" to the screen.
Recurse into first subfolder...

3.



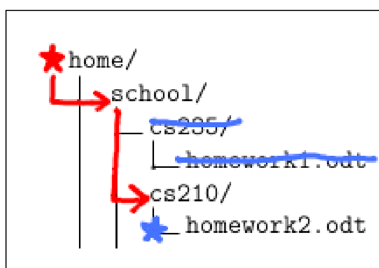
Recursive_DisplayFolder("cs235")
Write "cs235" to the screen.
No subfolders.
Display files.
Return.

4.



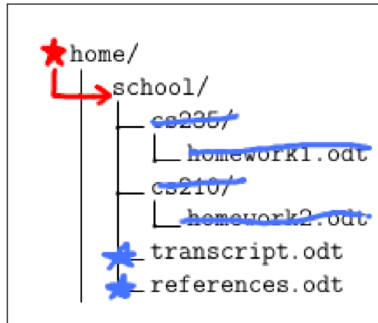
Recurse into second subfolder...

5.



Recursive_DisplayFolder("cs210")
Write "cs210" to the screen.
No subfolders.
Display files.
Return.

6.

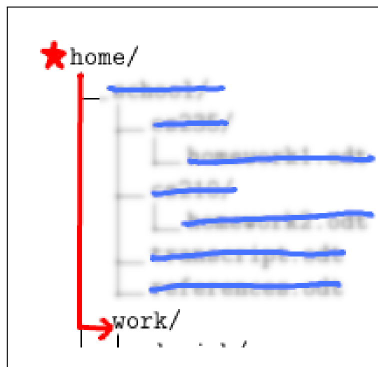


Finished with subfolders.
Display files.
Return.

7.

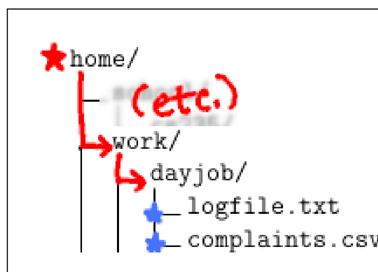
Recurse into second subfolder...

8.



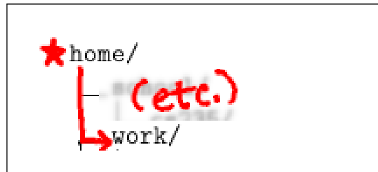
Recursive_DisplayFolder("work")
Write "work" to the screen.
Recurse into first subfolder...

9.



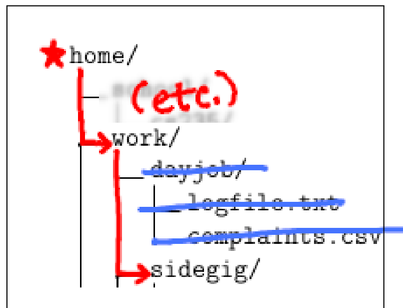
Recursive_DisplayFolder("dayjob")
No subfolders.
Display files.
Return.

10.



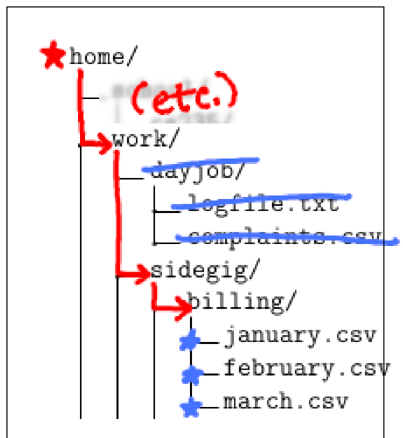
Recurse into second subfolder...

11.



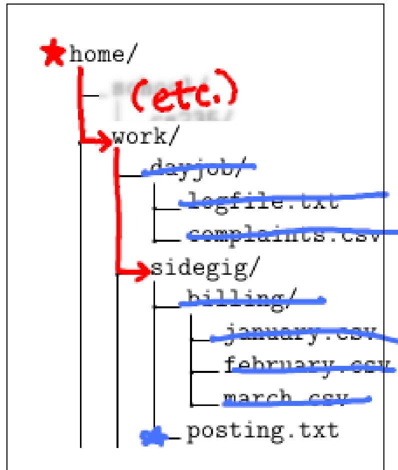
Recursive_DisplayFolder("sidegig")
Write "sidegig" to the screen.
Recurse into first subfolder..

11.



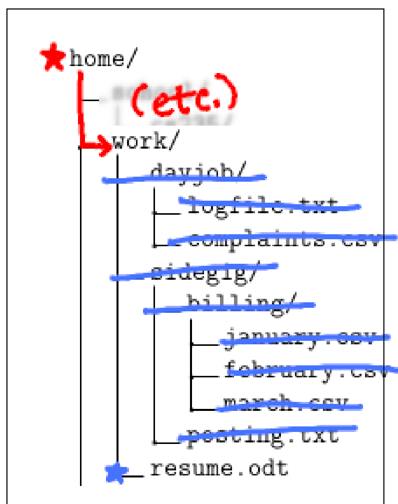
Recursive_DisplayFolder("billing")
Write "billing" to the screen.
No subfolders.
Display files.
Return.

12.



Finished with subfolders.
Display files.
Return.

13.

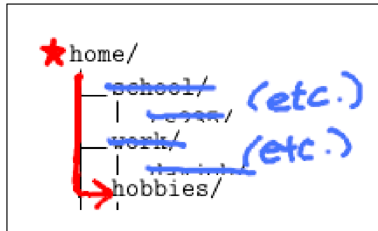


Finished with subfolders.
Display files.
Return.

14.

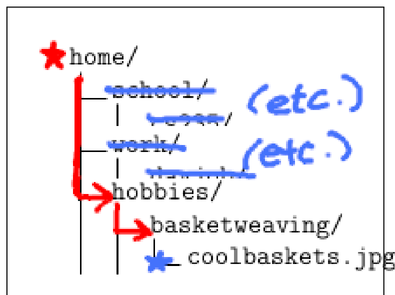
Recurse into third subfolder...

15.



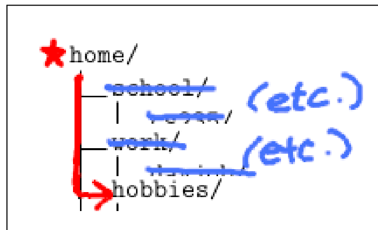
Recursive_DisplayFolder("hobbies")
Write "hobbies" to the screen.
Recurse into first subfolder...

16.



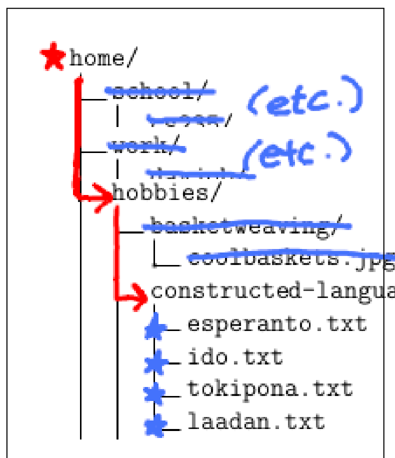
Recursive_DisplayFolder("basketweaving")
Write "basketweaving" to the screen.
No subfolders.
Display files.
Return.

17.



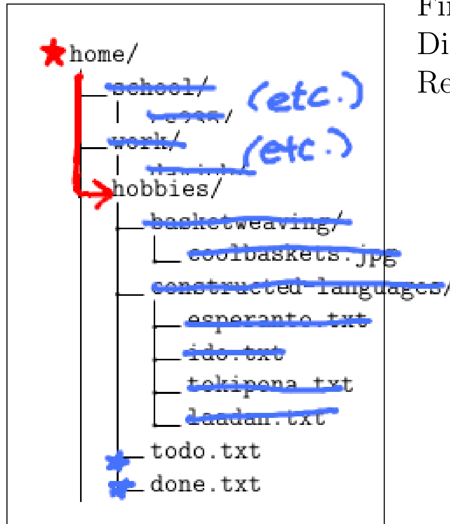
Recurse into second subfolder...

18.



Recursive_DisplayFolder("constructed-languages")
Write "constructed-languages" to the screen.
No subfolders.
Display files.
Return.

19.

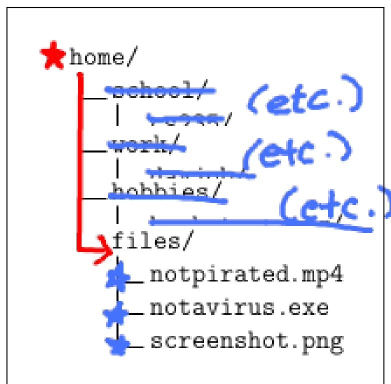


Finished with subfolders.
Display files.
Return.

20.

Recurse into fourth subfolder...

21.



Recursive_DisplayFolder("files")
Write "files" to the screen.
No subfolders.
Display files.
Return.

20.

Finished with subfolders.
Return.

21. DONE.

You can also study the **log.html** file to see each function's call and in what order:

FileSystemManager::Display	Display filesystem
FileSystemManager::Recursive_DisplayFolder	Display folder: home
FileSystemManager::Recursive_DisplayFolder	Display folder: school
FileSystemManager::Recursive_DisplayFolder	Display folder: cs235
FileSystemManager::Recursive_DisplayFile	Display file: homework1.odt
FileSystemManager::Recursive_DisplayFolder	Display folder: cs210
FileSystemManager::Recursive_DisplayFile	Display file: homework2.odt
FileSystemManager::Recursive_DisplayFile	Display file: transcript.odt
FileSystemManager::Recursive_DisplayFile	Display file: references.odt
FileSystemManager::Recursive_DisplayFolder	Display folder: work
FileSystemManager::Recursive_DisplayFolder	Display folder: dayjob
FileSystemManager::Recursive_DisplayFile	Display file: logfile.txt
FileSystemManager::Recursive_DisplayFile	Display file: complaints.csv
FileSystemManager::Recursive_DisplayFolder	Display folder: sidegig
FileSystemManager::Recursive_DisplayFolder	Display folder: billing
FileSystemManager::Recursive_DisplayFile	Display file: january.csv
FileSystemManager::Recursive_DisplayFile	Display file: february.csv
FileSystemManager::Recursive_DisplayFile	Display file: march.csv
FileSystemManager::Recursive_DisplayFile	Display file: posting.txt
FileSystemManager::Recursive_DisplayFile	Display file: resume.odt
FileSystemManager::Recursive_DisplayFolder	Display folder: hobbies
FileSystemManager::Recursive_DisplayFolder	Display folder: basketweaving
FileSystemManager::Recursive_DisplayFile	Display file: coolbaskets.jpg
FileSystemManager::Recursive_DisplayFolder	Display folder: constructed-languages
FileSystemManager::Recursive_DisplayFile	Display file: esperanto.txt
FileSystemManager::Recursive_DisplayFile	Display file: ido.txt
FileSystemManager::Recursive_DisplayFile	Display file: tokipona.txt
FileSystemManager::Recursive_DisplayFile	Display file: laadan.txt
FileSystemManager::Recursive_DisplayFile	Display file: todo.txt
FileSystemManager::Recursive_DisplayFile	Display file: done.txt
FileSystemManager::Recursive_DisplayFolder	Display folder: files
FileSystemManager::Recursive_DisplayFile	Display file: notpirated.mp4
FileSystemManager::Recursive_DisplayFile	Display file: notavirus.exe
FileSystemManager::Recursive_DisplayFile	Display file: screenshot.png

1.4 Recursive Find functions

There are two separate functions, one for finding files and one for finding folders, but they operate similarly. Each of these find functions have a public entry-point function:

```
File* FileSystemManager::FindFile( string name )
{
    Logger::Out( "Find file: " + name,
        "FileSystemManager::FindFile" );
    // Begin searching at root
    return Recursive_FindFile(m_allFolders[0], name);
}

Folder* FileSystemManager::FindFolder( string name )
{
    Logger::Out( "Find folder: " + name,
        "FileSystemManager::FindFolder" );
    // Begin searching at root
    return Recursive_FindFolder(m_allFolders[0], name);
}
```

And the function headers for each are:

```
File* Recursive_FindFile(
    Folder* ptrFolder,
    string name );

Folder* Recursive_FindFolder(
    Folder* ptrFolder,
    string name );
```

Where `ptrFolder` is the folder that we are currently inside of, searching, and `name` is the name of the file or folder we wish to find.

1.4.1 Folder* Recursive_FindFolder(Folder* ptrFolder, string name)

At the beginning of the function, we have a `ptrFolder` that we wish to investigate. We will need to see if *this* folder is the one we're looking for, and if it's not we will investigate each of its subfolders.

See the next page for hints.

1. Check to see if the `ptrFolder`'s name is the same as the `name` parameter. If it is, return `ptrFolder` as the folder we're searching for.
2. If we haven't returned yet, that means this `ptrFolder` isn't the one we're searching for. We will continue our search by searching each of `ptrFolder`'s subfolders. Create a **for loop** to iterate through all of its child folders (you can call the `GetFolders()` function that belongs to the `Folder` class). Within the for loop...
   (a) Create a pointer `Folder* result;`
   (b) Recurse into `Recursive_FindFolder( ptr, name );` using the current subfolder (see next page for hints), and store its return result in the `result` variable.
   (c) Check to see if `result` is NOT `nullptr`. Return `result` if its not nullptr.
3. At the very end of the function, outside of the for loop, we can assume that the folder has not been found in any of the subfolders. In this case, return `nullptr` to signify that the folder is not in this directory.

Running the function: When you run the program and search for a folder, it should return and display information about that folder (this part is already implemented in `main()`.) If not found, the Address returned is just 0.

```
 Enter folder name:

 >> hobbies

Address:            0x55b336d29860
Name:               hobbies
Type:               Folder
Total subfolders:   2
Total files:        2
```

Hints:

How do I get a folder's name?
The `Folder` class has a function called `GetName()` that you will use to compare its name to the parameter `name`. To access that function via the `ptrFolder` pointer, use `ptrFolder->GetName()`.

How do I iterate over its subfolders?
You can access the list of all this folder's subfolders via the `GetFolders()` function like this: `ptrFolder->GetFolders()`. This returns a `vector<Folder*>` object - an array of pointers to other Folders.

We can iterate over each of these subfolders with either a normal for-loop:

```
1 vector<Folder*> folders = ptrFolder->GetFolders();
2 for ( unsigned int i = 0; i < folders.size(); i++  )
3 {
4   // One folder is: folders[i]
5   Folder* result;
6   result = Recursive_FindFolder(folders[i], name);
7 }
```

or a range-based for-loop:

```
1 for ( auto& folder : ptrFolder->GetFolders() )
2 {
3   // One folder is: folder
4   Folder* result;
5   result = Recursive_FindFolder(folder, name);
6 }
```

Visualizing the search: On the next page we will look at the steps of how the recursive function is executed, searching for the Folder named **dayjob**.

1. ptrFolder = home "home" isn't "sidegig", so iterate over all the subfolders.	2. ptrFolder = school name doesn't match, recurse into subfolders.	3. ptrFolder = cs235 This has no subfolders, so our search ends here. Return.
4. ptrFolder = school We've returned to school, recurse into the next subfolder.	5. ptrFolder = cs210 This has no subfolders, so our search ends here. Return.	6. ptrFolder = school We've returned to school, but we're out of subfolders, so our search ends here. Return.
7. ptrFolder = home We've returned to home, recurse into the next subfolder.	8. ptrFolder = work name doesn't match, recurse into subfolders.	9. ptrFolder = dayjob this does match, so return dayjob.
10. ptrFolder = work return dayjob.	11. ptrFolder = home return dayjob.	Folder * dayjob returned! end of function; dayjob Folder pointer returned.

1.4.2 File* Recursive_FindFile(Folder* ptrFolder, string name)

This function operates similar to the FindFolder function, except that we aren't going to compare the current `ptrFolder`'s name to the `name` parameter, since that's a Folder and we're looking for a File. Instead, we're going to search the `ptrFolder`'s list of Files, and if we can't find the File, then we will recurse into its subfolders to search.

At the beginning of the function, `ptrFolder` is the Folder that we're currently searching in, and `name` is the name of the File we're looking for.

Again, see next page for hints.

1. First, we're going to look at all the files in the directory. Use a **for loop** to iterate through all the Files of `ptrFolder`. With the loop...
   (a) Check to see if the current File's name matches `name`. If it matches, return this File. Otherwise, do nothing.
2. At this point if we haven't hit the `return` statement within the previous for loop, the File is not within *this* Folder, so next we will search its subfolders. Use a **for loop** to iterate through all the Folders of `ptrFolder`. Within the loop...
   (a) Declare a `File* result;` object.
   (b) Recurse into `Recursive_FindFile( ptr, name );` using the current folder (`folders[i]` or `folder`). Store this function call's result in the `result` variable.
   (c) If the `result` is NOT `nullptr`, then return the `result`.
3. At the end of the function, if we haven't returned by now, that means we haven't found the File in this directory or any subdirectories. In this case, return `nullptr`.

Hints:

How do I iterate over its files?
Similar to how you'd iterate over its folders, except using the `ptrFolder->GetFiles()` function. This returns a `vector<File*>` object - an array of pointers to its Files.

We can iterate over each of these subfolders with either a normal for-loop:

```
vector<Files*> files = ptrFolder->GetFiles();
for ( unsigned int i = 0; i < files.size(); i++ )
{
  // One folder is: files[i]
}
```

or a range-based for-loop:

```
for ( auto& file : ptrFolder->GetFiles() )
{
  // One folder is: file
}
```

How do I check the file's name?
Depending on how you have written your loop, you will use something like...

```
if ( files[i]->GetName() == name )
```

or

```
if ( file->GetName() == name )
```

How do I return the file?
Depending on how you have written your loop, you will use something like `return files[i];` or `return file;`