CS303
Project 3

Gavin Stark
Alexandria Piatt

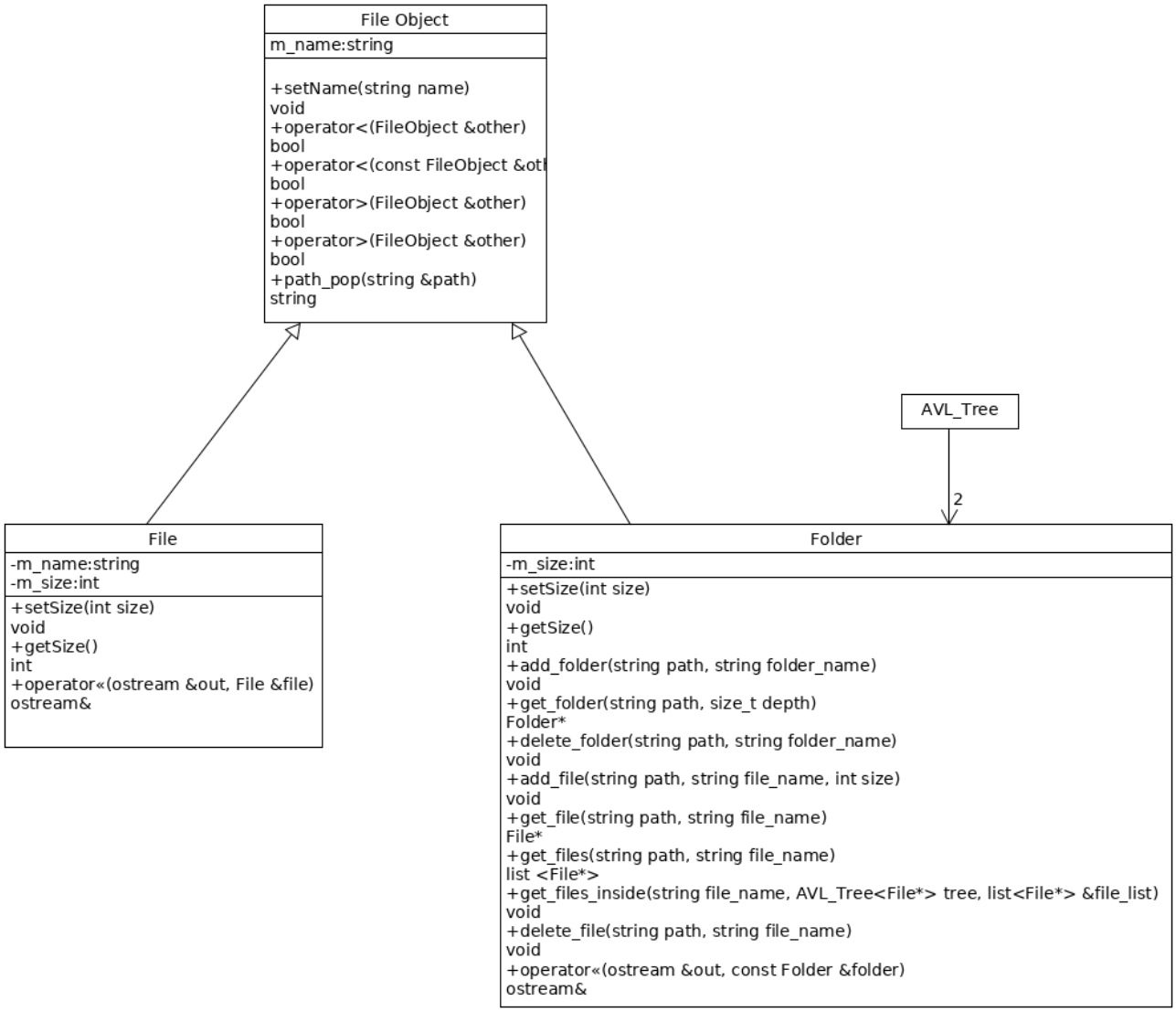
May 4th 2017

We did project 3A:

Assumptions:

- Assumed path would have ‘/’ to separate the folders names. (i.e. home/user/apgs/documents/here).
- Assumed all file sizes were the same magnitude (bytes).
- Assumed all added folders were empty when being added

UML Diagram


File Object
m_name:string
+setName(string name)
void
+operator<(FileObject &other)
bool
+operator<(const FileObject &ot
bool
+operator>(FileObject &other)
bool
+operator>(FileObject &other)
bool
+path_pop(string &path)
string
AVL_Tree
2
File
-m_name:string
-m_size:int
+setSize(int size)
void
+getSize()
int
+operator«(ostream &out, File &file)
ostream&
Folder
-m_size:int
+setSize(int size)
void
+getSize()
int
+add_folder(string path, string folder_name)
void
+get_folder(string path, size_t depth)
Folder*
+delete_folder(string path, string folder_name)
void
+add_file(string path, string file_name, int size)
void
+get_file(string path, string file_name)
File*
+get_files(string path, string file_name)
list <File*>
+get_files_inside(string file_name, AVL_Tree<File*> tree, list<File*> &file_list)
void
+delete_file(string path, string file_name)
void
+operator«(ostream &out, const Folder &folder)
ostream&

Efficiency of Algorithms

Folder::add_Folder(string path, string name)
The Big O is a combination of the Big O of the get_folder function (O(nlog(n)) for a full path traversal) * Big O(log(n)). The entire add_folder function has a Big O ((n+1)log(n)) or Big O(nlog(n)).

Folder::get_folder(string path, size_t depth)
In best case scenario (if the given path is the root folder) is O(1). If it has to iterate through the tree it is O(log n) for a single piece of the path. It is O(nlog(n)) for the entire path.

Folder::delete_folder(string path, string name)
First the function calls get_folder which has a typical Big O (nlog(n)). When combined with the erase function, the final Big O(nlog(n)) for the entirety of delete_folder.

Folder::get_file(string path, string file_name)
Again, the function calls get_folder with a standard Big O(nlog(n)). Then it searches the AVL_Tree that holds the files in the given folder. Searching that tree is also Big O(nlog(n)) to find the specific file so all together the efficiency is Big O(nlog(n))^2).

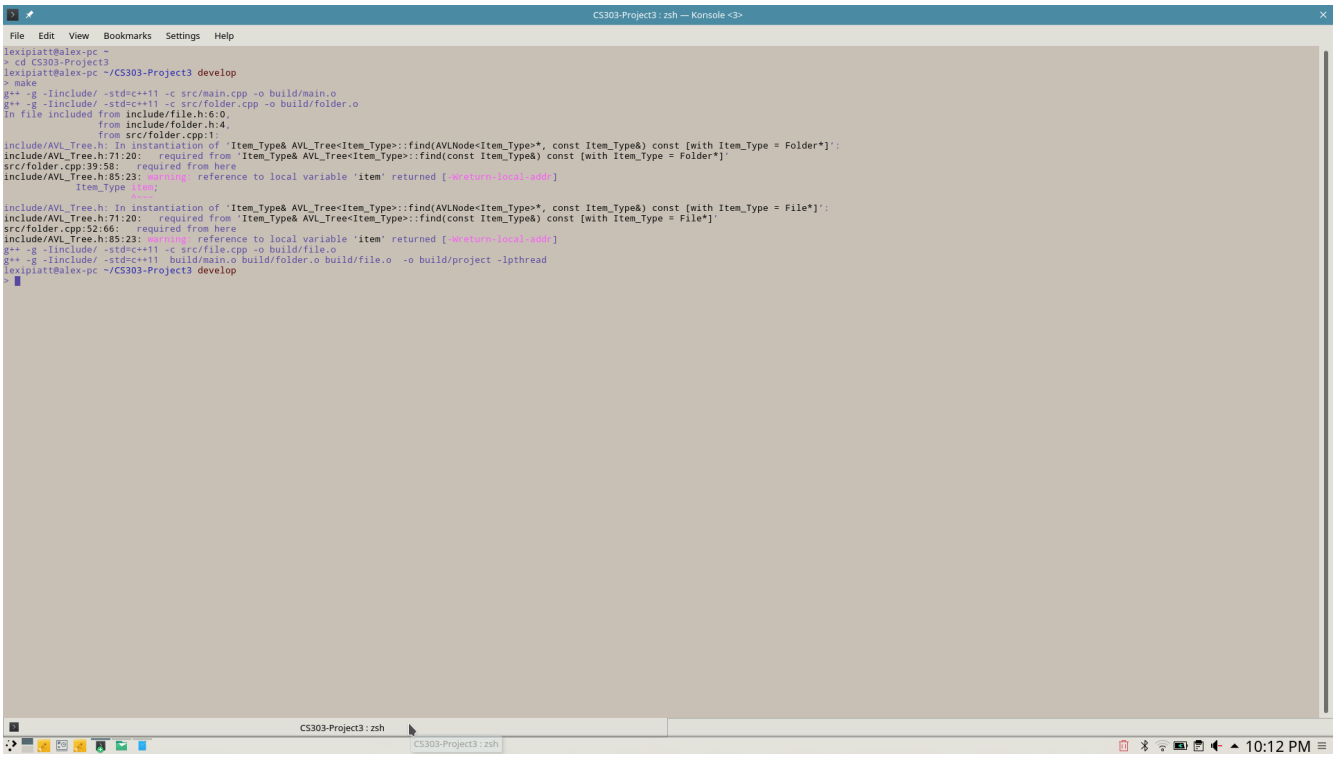
Folder::add_file(string path, string name, int size)
The Big O is a combination of the Big O for get_file and the Big O of the insert function. It increments the size of the files at the same time as the insertion, so the Big O is unaffected. The final Big O of the function will be Big O(nlog(n)).

Folder::delete_file(string path, string name)
The Big O is Big O(nlog(n)) which is the combination of the get_file and delete functions. It changes the size as it goes so, the big O is unaffected.

Folder::get_files(string path, string file_name)
get_files wraps the get_files_inside function, so its Big O is dependent on the get_files_inside. On its own get_files is a O(1) function, but with the call to get_files_inside makes the Big O(n).

Folder::get_files_inside(string file_name, AVL_Tree<File*> tree, list<File*> &file_list)
The Big O of this file is O(n) because it has to traverse and visit every node in the tree.

Compiling Screenshot



References:

Linux Man Pages(find a copy here: https://linux.die.net/)
http://www.cplusplus.com/reference
https://www.tutorialspoint.com/cplusplus
Kuhail's Source Code Used (AVL_Tree.h and AVL_Node.h)