Advanced Programming

C1057352

Introduction:

In this report I will discuss the key areas of competencies in my Advanced Programming project. The main areas discussed will be sensible code separation, adherence to object-oriented programming (OOP) in C++, and dynamic memory management. Each section assesses my approach and will use the data presented in the figures to justify the self-assigned grades based on the outlined criteria.

Sensible Code Separation (Grade: 14/16):

My code architecture features well-defined classes such as Directory, File, and FileSystemComponent, ensuring that attributes and properties remain within their respective objects. This ensures that each component's responsibilities are encapsulated appropriately. The use of a std::vector<std::pair<std::string, std::unique_ptr<FileSystemComponent>>> to manage file and directory entities within a single container is a testament to efficient data management, minimizing space complexity by avoiding multiple containers.

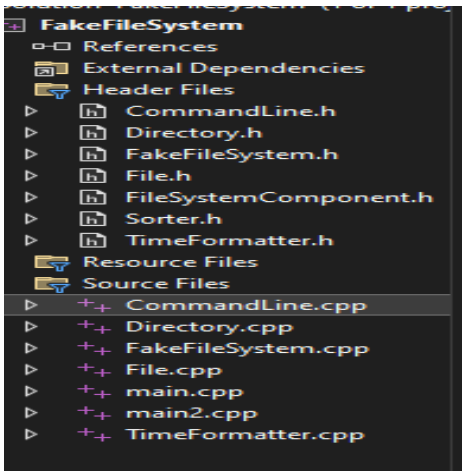

Figure 1: Visual representation of class separation and data organization.

As is illustrated by figure 1, the code is organized with the intent of splitting the tasks into micro tasks which has allowed for a smoother transition in the development process. While also categorizing the code based on functionality with-in each class. This justifies my self-assessment of a 14/16, as this is a testament reflecting my proficiency in applying sensible code separation to enhance the readability and maintainability of the code.

Use of OO in C++ (Grade: 15/16)

In aligning with the project's specifications, I refrained from using getters and setters, an atypical but deliberate deviation from common OOP practices. My commitment to OO principles was further exemplified by integrating Google's testing framework for C++ demonstrates a deep understanding of OOP, enabling robust unit testing and validation of class behaviors.

```
Running main() from D:\a\_work\1\s\ThirdParty\googletest\googletest\src\gtest_main.cc
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from FakeFileSystemTest
[ RUN      ] FakeFileSystemTest.ChangeDirectoryTest
[       OK ] FakeFileSystemTest.ChangeDirectoryTest (0 ms)
[ RUN      ] FakeFileSystemTest.CreateDirectoryTest
[       OK ] FakeFileSystemTest.CreateDirectoryTest (0 ms)
[ RUN      ] FakeFileSystemTest.CreateFileTest
[       OK ] FakeFileSystemTest.CreateFileTest (0 ms)
[----------] 3 tests from FakeFileSystemTest (2 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (5 ms total)
[  PASSED  ] 3 tests.
```

Figure 2: System testing results using Google's framework.

```
Running main() from D:\a\_work\1\s\ThirdParty\googletest\googletest\src\gtest_main.cc
[==========] Running 4 tests from 2 test cases.
[----------] Global test environment set-up.
[----------] 2 tests from FileTest
[ RUN      ] FileTest.ListContentTest
[       OK ] FileTest.ListContentTest (0 ms)
[ RUN      ] FileTest.IsSmallerThanTest
[       OK ] FileTest.IsSmallerThanTest (0 ms)
[----------] 2 tests from FileTest (3 ms total)

[----------] 2 tests from DirectoryTest
[ RUN      ] DirectoryTest.AddEntityTest
[       OK ] DirectoryTest.AddEntityTest (0 ms)
[ RUN      ] DirectoryTest.DeleteEntityTest
[       OK ] DirectoryTest.DeleteEntityTest (0 ms)
[----------] 2 tests from DirectoryTest (2 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 2 test cases ran. (10 ms total)
[  PASSED  ] 4 tests.
```

Figure 3: Unit testing results using Google's framework.

My experience with C#MSTest made the development simpler. Figure 2 and 3 highlight how far I went in the development of the application ensuring that the classes were not only encapsulated but also worked for their intended purposes. This justifies my self-assessment grade of 15. This

grade reflects a high level of mastery in applying OO principles and test-driven development methodologies.

Dynamic Memory Management (Grade: 12/16)
Initially, my project utilized smart pointers only. However, to demonstrate deeper understanding of manual memory management, I transitioned to direct heap allocation and deallocation for some of the sections like root, current directory. This allowed me to showcase my ability to manage memory without automated tools, though it introduced complexities that highlighted areas for potential improvement.

```
private:
    std::vector<std::pair<std::string, std::unique_ptr<FileSystemComponent>>> content;
    Directory* parentDirectory;
```

Figure 4: Smart pointers implementation memory management.

```
FakeFileSystem::FakeFileSystem(string rootPath) : root(Directory(rootPath, std::time(nullptr))) {
    currentDirectory = &root;
    initialize();
}

FakeFileSystem::~FakeFileSystem() {
    delete& root;
}
```

Figure 5: Manually deleting the root, in this scenario i did not need to delete the current directory as it is automatically deleted when root is deleted due to being a pointer.

Although my methods worked and I was able to get the task done. I Acknowledge that there were challenges and learning opportunities that presented themselves in that section, as this was my first time dynamically handling memory. Due to some mistakes that were recognised later during the project I was unable to fix the parentDirectory, which should have been a weak smart pointer. This grade reflects competent handling of dynamic memory with an understanding of its implications on software reliability and performance.

Conclusion
I want to further clarify that there were several other development approaches I took that further support my justification of awarding myself these grades, such as figure 6. The usage of the template for sorting highlights that I have followed the Don’t Repeat Your self concept. Which makes the code maintainability sign significantly better especially as the application becomes more complex.

Figure 6: Template for the sorting algorithm used for sort by alphabetical and sort by size.