Studio 20: CSE 332S File System Assignment Part 5

Key Topic: Structural Design Patterns

Last studio, we used the visitor design pattern to add the ability to read files and display them in ways other than described in the read() function of the concrete file classes. This studio, we will introduce the functionality to add password protected files to our filesystem. We will use the proxy pattern to accomplish this.

Unit tests have been provided for you to test your code before moving on to the next part of the assignment. ***You must follow the naming conventions (highlighted in blue) described in the instructions below in order to pass the tests!***

All of the following parts must be completed.

1. **Do not continue unless you have completed Studio 19, complete Studio 19 instead.** Use Zoom to create a meeting and invite your group members as in studio 16. I would suggest only 1 group member writes code during the studio, ideally a different team member than the last studio. At the end of the studio, you will commit and push those changes so that the other group members can pull the changes from the shared repository and merge them into their own local repository. Review the git workflow information from Studio 16 if needed.

   Open "*Studio20.sln"* and switch to the "Solution Explorer" tab. Open the provided ReadMe.txt file and in it, record your answers as you work through this studio. For the answer to this question, record all team members' names.

   Note: All of the header and source files you will modify throughout studios 16-21 and lab 5 are stored in a common location, the SharedCode folder inside of your repository. All of the 6 studio solutions link to the same files, so when you edit a file in studio 16, those files will be updated for studio 17 as well. No setup should be necessary. One side effect of this is that as you modify files in later studios, some tests from previous studios will fail or break. This is ok. You do not need to worry about the tests from previous studios once that studio is complete.

2. We will use the Proxy pattern to allow files to be password protected. When a file is created that is password protected, a file proxy should be added into the file system so the proxy object points to the original file.

To start, create two new files, “PasswordProxy.h” and “PasswordProxy.cpp”. *Be sure to create these files in the SharedCode folder of your repository.* Within these files, we will declare and define a new class, PasswordProxy.

PasswordProxy should publicly inherit from the AbstractFile class. Declare and define the following:

a. A private pointer to an AbstractFile - this will point to the file that is to be password protected
b. A private string - this will be the password for the real file
c. A constructor that takes in a pointer to an AbstractFile and a string representing the password for that file - you should set the member variables accordingly using the base/member initialization list
d. The destructor that deletes the pointer to the real file
e. A protected method passwordPrompt that returns a string - this method should prompt the user to input a password and returns the password entered by the user

You may also want to create a protected helper function that checks to see whether a string matches the password member variable to help you later on.

As the answer to this question, explain why it is necessary to delete the pointer to the actual file in the PasswordProxy destructor.

3. The PasswordProxy should also inherit all of the public functions declared in the AbstractFile class as follows:
    a. read - the function should prompt the user for a password using the passwordPrompt method, and check if the password is correct. If the password is correct, the function should return the results of calling read on the actual file, or else it should return an empty vector
    b. write - the function should prompt the user for a password using the passwordPrompt method, and check if the password is correct. If the password is correct, the function should return the results of calling write on the actual file, or else it should return a unique non-zero error value
    c. append - the function should prompt the user for a password using the passwordPrompt method, and check if the password is correct. If the password is correct, the function should return the results of calling append on the actual file, or else it should return a unique non-zero error value
    d. getSize - the function should return the results of calling getSize on the private member variable
    e. getName - the function should return the results of calling getName on the private member variable

f. accept - the function should prompt the user for a password using the passwordPrompt method, and check if the password is correct. If the password is correct, the function should call accept on the actual file

4. In main(), dynamically allocate an AbstractFile of your choice and a PasswordProxy of that file (make the password whatever you would like). Try reading, writing, and displaying the file via the proxy object.

   As the answer to this part, describe what tests you performed and what results you observed.

5. Finally, we will be posting unit tests for you to test your code on Piazza. They will be posted in the form of a text file, so you will need to copy and paste the tests appropriately into the UnitTest.cpp file in your test solution.

   Next studio, we will create a command prompt object that maintains a file system object and a file factory object as well as a set of “command” objects that allow the user to interact with the file system!