Studio 16: CSE 332S File System Assignment Part 1

Key Topic: Object-oriented programming principles

Throughout the remaining studios and lab 5, you will be building a simulated file system and a terminal to interact with the file system. The file system will contain several different file types (text files, image files, etc.). In this studio, you will create an abstract base class to declare the interface that contains behaviors common to all types of files. you will then define a concrete class that implements the interface, defining a single specific file type.

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!***

Repository setup and suggested studio process:

Form a team, create, and clone your repository:
From this studio on (through lab 5), you will be working with the same group. A group may have 1-3 members. Please make use of piazza to recruit team members if needed. **Before continuing**, make sure you have communicated with and decided on your group members. One member of your group should visit the following link:

Create team and repo here

to create a team and repository for your group. Distribute the team name to the remaining team members. The remaining team members should then visit the link above and join the existing team. At this point, you should all have access to the shared repository.

Each team member should clone a local copy of your repository from the “Team Explorer” tab within Visual Studio, as in studio 2. **You should not modify the file structure of the repository in any way.** We have already created Solutions for each of the studios and populated them with the necessary files.

Suggested studio process:
(If in class use a breakout room in your TAs zoom meeting) One group member should use Zoom to create a meeting and invite (or share the zoom link with) all other group members so that you may work together on the studio. I would suggest only 1 group member writes code during the studio, sharing their screen using the screenshare feature in Zoom so the entire group can follow along. When you are finished working on the studio, the coder will then commit and push their code to your shared remote repository. All other members can then do a “pull” to pull changes from the shared repository and merge them into your local copy. Using git while

working remotely, you will all have your own local copy of the repository that you will make changes to, however you will share a common remote repository. It is important to keep your local repository up to date with the remote repository (pull) and the remote repository up to date with your changes (commit and push) to avoid conflicts between the two. I would suggest that each time you begin work on a studio or the lab, you first issue a pull request to ensure your local copy of the repository is up to date with the remote. After you finish working, commit and push all changes you make so that your teammates will be able to pull those changes, keeping their own repository up to date. For more details and a better understanding of how git works, please look over and think through this tutorial. The tutorial is written to be worked on using a terminal and issuing git commands via the command line, however the concepts and commands discussed are helpful regardless and the same commands are available in Visual Studio via the "Team Explorer" tab.

If you end up with a repository containing many conflicts between all of your local repositories, it will be tedious and time consuming to resolve. Spending time to review the git tutorial above and ensuring you always pull before starting work as well as commit and push your changes when you finish will save you time in the long run.

All of the following parts must be completed.

1. Within Visual Studio, open the "Team Explorer" tab. From the "manage connections" screen, make sure your repository is listed under "Local Git Repositories". If it is not, click "Add", navigate to your cloned copy of the repository, and open it. You should see the repository appear under "Local Git Repositories". Double-click it to make it your active repository. All studio solutions have already been created for you. You should see them listed at the bottom of the"Home" screen of the "Team Explorer" tab. Open "*Studio16.sln"* and then switch to the "Solution Explorer" tab. You should see the "Studio16" project as well as "UnitTest". All of your code should be written within the "Studio16" project. "UnitTest" contains a set of test provided so that you may test your codes' functionality. How to run the tests is described at the end of this studio in step 6. Open the provided ReadMe.txt file (in the Studio16 project) and in it, record your answers as you work through this studio. For the answer to this question, record all team members' names.

2. A file system controls how data is stored and later retrieved. File systems typically store data in chunks called "files". In our file system, we will support several different file types; more importantly we should be able to easily add support for new file types as needed. To support this flexibility, we will program our client code that interacts with file objects to use a common interface that is supported by all file types. This will allow the client code to work with an object of any concrete file class that defines the shared interface. So, if the client code is the file system, the file system will be able to store and manage files of many different types. In this studio we will create an abstract base class

that declares the interface of a file, then we will a concrete class that inherits and defines the interface in a way specific to Text Files.

a. As the answer to this question, describe the process of declaring an “interface” in C++.

Now, open “AbstractFile.h”. In this header file, declare the “AbstractFile” interface. An “AbstractFile” should contain the following public member methods (**again, please be sure to name your functions as you see them below**):

- read - read takes no arguments and returns void
- write - write takes a single parameter of type std::vector<char> and returns an int
- append - append takes a single parameter of type std::vector<char> and returns an int
- getSize - getSize takes no parameters and returns an unsigned int
- getName - get Name takes no parameters and returns a std::string

3. The first concrete file class we will create will be a text file. You will declare and define this class in “TextFile.h” and “TextFile.cpp”. In TextFile.h, declare a new class called “TextFile”. TextFile should inherit publicly from AbstractFile.

   a. Is this an example of “interface inheritance” or “implementation inheritance”?

   The TextFile class should have 2 member variables: a std::vector<char> that will hold the files contents and a std::string that will hold the name of the file.

   b. Should these member variables be public, protected, or private?

   Declare and define a constructor for the TextFile class that takes a single parameter, a std::string, and uses it to initialize the name member variable of your class.

   Define the interface inherited by your Abstract File class as follows:

   - getSize returns the size of the vector<char> member variable
   - getName returns the std::string member variable
   - write updates the vector<char> member variable to be a copy of the vector<char> parameter passed to write. Anything previously in the member variable is overwritten. Write should return 0 if the write was successful. Later as we add additional file types to our program, a write may fail and will return a non zero value.
   - append adds the contents of the vector<char> parameter to the end of the vector<char> member variable representing the contents of the file. For now, return 0 meaning the append was successful.
   - read loops over the contents of the file and inserts each char of the vector<char> to cout

4. In the main function of your program (Studio16.cpp), create a variable of type “TextFile”. Write to the file and try reading from the file as well as other tests to ensure the Text file class is working as expected. As the answer to this question, describe the test cases you ran and if your TextFile class behaved as expected.

5. In question 4, we restrict main to interact with a TextFile only. Update main to interact with an arbitrary file type rather than a single concrete file type.
    a. As the answer to this question, describe how this was accomplished. You can copy and paste your main function here as well.

    Rerun your tests from step 4 and ensure your code still behaves as expected.

6. Finally, test your code using the given unit tests to ensure you are ready to move on to the next part of the File System assignment.

    To test your code perform the following steps:
    1. Make sure you have built your program.
    2. Select “Test >> Windows >> Test Explorer” from the top bar of your VS window. This will open the Test Explorer Window.
    3. Click the green arrow to run ALL tests. Failed tests will show up red and passed tests will show up green.
    4. Go back and fix any code that did not pass the given tests! Look at the test for more details on what might have gone wrong.
       Note: Rebuild after you make any changes to your solution before you rerun the tests!

Make sure to commit and push your work. All other group members should pull to merge changes into your own local copy of the repository.