Cpt S 422 HW 12 – "HW5 Redux: ConcatStream and Tests"

Read all instructions *carefully* before you write any code.

Overview

This assignment is very similar to HW5 in terms of content and requirements. It also will be graded in a way that allows you to potentially earn back points on HW5. But you must adhere to the requirements, make use of the online automated feedback, and submit correct, working code BEFORE the due date/time. There will be no additional attempts permitted beyond the 5 submissions allowed by the server. Read the remaining details very carefully about the grading and policies. After that, the implementation requirements are listed and must be read carefully as well.

Like many of the previous coding assignments, you will earn a score out of 20 points on this assignment. This score will be your HW12 grade, regardless of your performance on HW5. In other words, even if the score is LOWER than your HW5 grade, it WILL be your HW12 grade. In contrast, your HW5 score will become Max(Current_HW5_Score, HW12_Score) after the grading of this assignment. So if you were to receive 100% on HW12, you will also receive 100% on HW5, regardless of the previous score.

You will submit your code to the EECS server at http://eecs.wsu.edu/~eolds/422/index.html. The server has had its primary issues addressed and it should remain up (minus perhaps small maintenance tasks, which I will announce if need be) during the next week. The assignment is due on **Friday, November 18, by 11:45 PM**. This is going by the server clock, so if your clock is slow, the submission will seemingly cut off before the due/date time. You have been warned and no submissions will be accepted late, regardless of whether or not YOUR clock was still reading before the due date/time when you submitted. Submit at least an hour early to avoid issues. No submissions will be accepted after the server submission closes.

An additional note about submitting through to the EECS server, is that it has strict blocking policies and it's possible that some off-campus IP addresses are blocked. Very few students have had problems submitting (when the server is up) but a few of you have. Come to campus to submit your code if need be, since campus-issued IP addresses are less likely to be blocked. Obviously, also do this well before the due/date time. Contact me ASAP should you not be able to access the EECS page even from on campus.

You must look up and use the feedback from the EECS site grader for each and every submission you make. There will be no resubmissions granted, even if you got a 0 for a small mistake, if you did not read the feedback and address the issue(s) appropriately. If it said your code did not compile when you've ensured that it DOES compile on your Linux machine, then contact me ASAP. This is probably the only situation where I can allow you to have an additional submission, provided it's still before the due date, once we determine why it compiled on your machine but not the server.

The server-side grading app will use your code much like it would be used in practice. If tests fail, it implies that there is an issue with your code and YOU will be responsible for finding it. The feedback will not say "you've got an error on line 68 because you did [this] and it should be [this]". It will instead say something like seeking failed, data was corrupt at a particular position, length was wrong, and so on. This means you will need to investigate the issue and probably develop more unit tests. When you get feedback that there is a problem with your code, do not "guess" as to what's wrong or hastily implement a solution and resubmit. Take the time to properly investigate and produce a good fix. You will NOT be given more than 5 submission attempts for any reason.

Elaborating on the testing that you must do after getting feedback from the server, it's worth first pointing out that you should know exactly how a stream is supposed to work. That's a concept from a prerequisite course and it was also reviewed during the beginning of this course. It's also the case that if you put a buggy stream into use in an application in the real world then you'd likely get some sort of "weird output" and would be responsible for tracking down issues. You don't often get a simple message telling you exactly what's wrong and where the issue resides in the real world of software engineering. For this reason, it's worth devoting time to further develop testing skills. This is just as much, if not more, of a focus in this assignment as is the actual logic behind the ConcatStream class.

With respect to developing the skill of finding and fixing bugs, there's no more effective way of doing this other than spending your time writing code and tests. This assignment is having you do exactly that. There's no magic lesson that you have yet to receive in academics that will allow you to find bugs more easily. You just need to put in the time and practice coding to develop the skill. If you are unable to work out test cases to find issues, then you are likely lacking understanding of some fundamental concepts and need to seek help. Needless to say, you also need to seek out this help some time well before the day the assignment is due so that you have time to implement the code properly after gaining better understanding of the concepts.

Implementation Requirements

In this assignment you will (re-)implement the ConcatStream and NoSeekMemoryStream classes. <u>You will also implement unit tests that will be submitted with your assignment zip file.</u> You must strictly adhere to the given specification and write extensive unit tests to ensure this adherence.

Implement the ConcatStream Class

Implement the ConcatStream class in the CS422 namespace. Implement it in the ConcatStream.cs file. Remember to match the naming exactly, including casing. Also be sure to include all declarations necessary for proper ConcatStream functionality in the ConcatStream.cs file. It must be the case that you can compile a class library from this file alone with no errors using:

```
mcs -target:library ConcatStream.cs
```

This also applies to the NoSeekMemoryStream class that you will implement afterwards.

Important: even though your web server will only use the ConcatStream as a read-only stream, you must implement both the Read and Write functions for this assignment. We are developing a class that has utility even outside of the context of how it would be used by the web server. Both the Read and Write functions must throw an exception if either of the 2 streams does not support that action. The writing functionality will be described later.

For the most part, the functionality of the ConcatStream is limited by the lesser-functional of the two streams being concatenated. So the ConcatStream can read only if both streams can read and write only if both streams can write. But there is also some functionality that can be achieved potentially if one of the 2 streams does not support it. Implement two constructors in this class:

```
ConcatStream(Stream first, Stream second)
ConcatStream(Stream first, Stream second, long fixedLength)
```

The first constructor takes two Stream objects, and only supports the "Length" property if both streams support it. The first stream must support querying the length, but the stream is permitted to not have a length. If the first of the two does not have a length, then it's not possible to know where it ends and the next stream begins, so in this case throw an ArgumentException. The ConcatStream cannot be implemented without knowing the length of the first stream.

The second stream, on the other hand, should be allowed to be a stream without a length or the capability to seek, such as a NetworkStream. If the second stream doesn't support seeking, have the ConcatStream provide forward-only reading and writing functionality with no seeking.

Much of the above applies to the second constructor as well, only this time even if the second stream does not have a length, you will support the Length property. Have the second constructor store the fixed length value and return it in the getter for the Length property. You'll also need some way of storing information about which of the two constructors was used to instantiate the class, since the Length property behaves differently based on how the ConcatStream is constructed.

When the ConcatStream is constructed (using either constructor) its position must be at 0, regardless of the position of the first stream when it was passed to the constructor. If the second stream does not support seeking, assume it's at position 0 when passed into the constructor. It should have been implied from requirements listed prior, but it's worth pointing out that you'll need a member variable in the ConcatStream class to keep track of the position within the stream.

The writing functionality will be such that it never expands the length of the first stream, but it can overwrite existing contents in that stream. As an example, suppose you have 2 writable streams, the first of which is 20 bytes in size and currently at position 10 (meaning the ConcatStream's position is also at 10), and you write 30 bytes from a buffer. Since there are only 10 bytes left in the first stream and you cannot expand that stream, you write the first 10 of 30 bytes to it. Then you write the remaining 20 to the second stream. The ConcatStream must also be able to expand provided the following two things are true:

1. The second of the two streams supports expanding
2. The 2-parameter constructor was used to instantiate the ConcatStream (i.e. the stream is not fixed length due to the use of the 3-parameter constructor)

Do NOT let the stream expand if the 3-parameter constructor was used, even if the second of the two streams supports expanding.

If a write call cannot be completed then throw an exception, but only do so if the action cannot possibly be completed. For example, if your ConcatStream position is 32 and the first stream has a length of 20, then you could only write correctly to the second stream if its current position was exactly 12. In the case when the second stream supports seeking, it's easy enough to ensure that, so just seek to the appropriate spot and write in those circumstances. But if the second stream doesn't support seeking, then only complete the write if you are at the exact correct position. Otherwise throw an exception.

Seeking

```
public override long Position
public override long Seek(long offset, SeekOrigin origin)
```

Your stream must support seeking properly through use of both the Position property and Seek function. Make sure your unit tests include use of both of these. You may assume that the position will not be set to a negative value, but that's the only assumption you may make. You need to read the documentation to determine how to properly implement all other cases of seeking.

Length

```
public override long Length
```

Your stream must support querying length in all cases where it can be accurately determined. If you construct a ConcatStream with the 3-parmeter constructor, it is of fixed length and the length passed to that constructor must always be returned by the Length property for that instance. If the 2-parameter constructor was used, then you must return the combined length of the two streams. Should either one throw an exception when you query its length in this case, it's fine to let that exception bubble-up to outside of that function.

Implement the NoSeekMemoryStream Class and Tests

Since you'll want to test the ConcatStream with a second stream that does not support seeking, you will need such a stream to use in unit tests. Network streams are good, since they are ultimately what we will be using as the second stream in our web server, but they are excessive for a set of simple unit tests. Therefore, you will implement the NoSeekMemoryStream that behaves just like a memory stream, but does not support seeking.

Implement the NoSeekMemoryStream in the CS422 namespace and in the NoSeekMemoryStream.cs file. Make it inherit from MemoryStream and start with the declaration below, filling in the missing pieces as needed.

```
/// <summary>
/// Represents a memory stream that does not support seeking, but otherwise has
/// functionality identical to the MemoryStream class.
/// </summary>
public class NoSeekMemoryStream : MemoryStream
{
   public NoSeekMemoryStream(byte[] buffer) // implement

   public NoSeekMemoryStream(byte[] buffer, int offset, int count) // implement

   // Override necessary properties and methods to ensure that this stream functions
   // just like the MemoryStream class, but throws a NotSupportedException when seeking
   // is attempted (you'll have to override more than just the Seek function!)
}
```

Write a set of unit tests for your ConcatStream class. Include some unit tests that use a NoSeekMemoryStream as the second stream and make sure that your ConcatStream supports forward reading through all the stream contents.

Write additional unit tests as you see fit for ConcatStream as well as a few for the NoSeekMemoryStream class itself. Include the following and remember to submit all unit test code as well as the ConcatStream and NoSeekMemoryStream code:

- ConcatStream unit test that combines two memory streams, reads back all the data in random chunk sizes, and verifies it against the original data
- ConcatStream unit test that combines a memory stream as the first and a NoSeekMemoryStream as the second, and verifies that all data can be read
- ConcatStream tests that query Length property in both circumstances where it can and cannot be computed without exception.
- NoSeekMemoryStream unit test that attempts to seek using all relevant properties/methods that provide seeking capabilities in a stream, and makes sure that each throws the NotSupportedException.

Your unit tests can either be NUnit-based tests that run in MonoDevelop or standalone terminal application. Implement unit tests in different code files than the stream classes. You code must be able to compile in Linux with any of the following commands:

```
mcs -target:library ConcatStream.cs
mcs -target:library NoSeekMemoryStream.cs
mcs -target:library ConcatStream.cs NoSeekMemoryStream.cs
```