

WinForms “Notepad” Application / Fibonacci BigInt Text Reader

Cpt S 321 Homework Assignment

by Evan Olds

Submission Instructions:

Make a ZIP file that contains all source code files and the Visual Studio project and solution files, but not compiled executables. Submit to Blackboard Learn (<http://learn.wsu.edu>) BEFORE the due date/time. In the unlikely event that Blackboard is down, email the zip to the TA BEFORE the due date/time. You may optionally include a readme.txt file in the zip with any relevant information that you want the grader to be aware of.

Assignment Instructions:

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

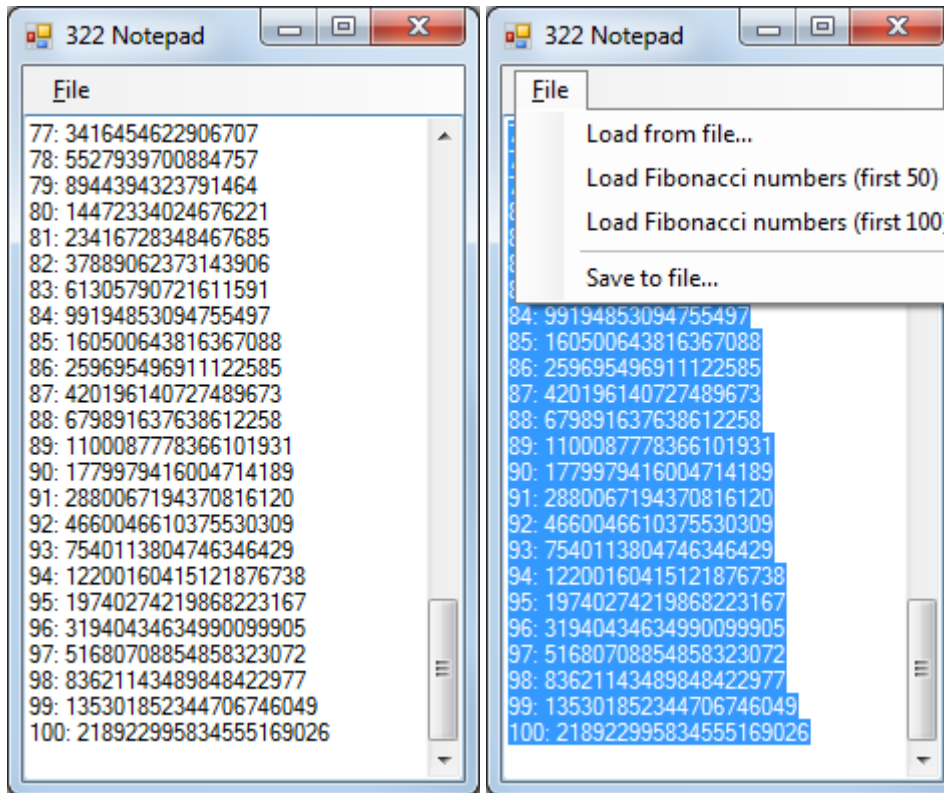
Using Visual Studio or SharpDevelop, create a WinForms application that can load and save plain text files. This will be much like a simplified version of the Notepad utility application that comes with Microsoft Windows. This app will also include the ability to load text from a non-file source (Fibonacci in-memory sequence generator).

Interface:

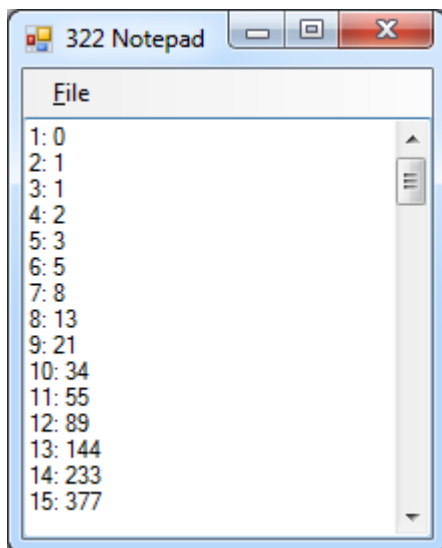
You are free to design the interface the way you wish for the most part, but you must include the following:

- a prominent text area that contains the text
 - Use a TextBox control
 - Set the Multiline property to true
 - Set the ScrollBars property to Vertical or Both so that vertical scrolling can occur
- buttons or menu options to load and save files
 - You will need 3 options for loading text, one from file, two from another source. This is discussed more below.
- use [OpenFileDialog](#)/[SaveFileDialog](#) for selecting files to open/save
 - Can be created in code or at design time by dragging them in from the toolbox

The image below shows an example of what the interface could look like:



Here's another image of the application scrolled up to the beginning numbers:



Saving Functionality (1 point):

Your application needs to provide the ability to save the text to a text file. Provide a button or menu option that brings up a [SaveFileDialog](#), prompting the user for a file name. If the user clicks OK in that dialog, which you'll determine by checking the return value of the `SaveFileDialog.ShowDialog` function,

then save the text in the text box to that file. You can save it using streams, text-writers, or any other method you see fit (it's the loading that's going to be more specific).

Loading Functionality (8 points total):

1. Generic Loading Function (1 point)

Write a function called "Load" or "LoadText" in your main form's .CS file. This will take a System.IO.TextReader object as a parameter. It should read all the text from the TextReader object and put it in the text box in your interface.

- a. Declare and implement the method mentioned above:

private void LoadText(TextReader sr)

- b. Use either [ReadToEnd](#) or [ReadLine](#) (in a loop) to load all text from the reader and put it in the text box. This should replace any content already in the text box.

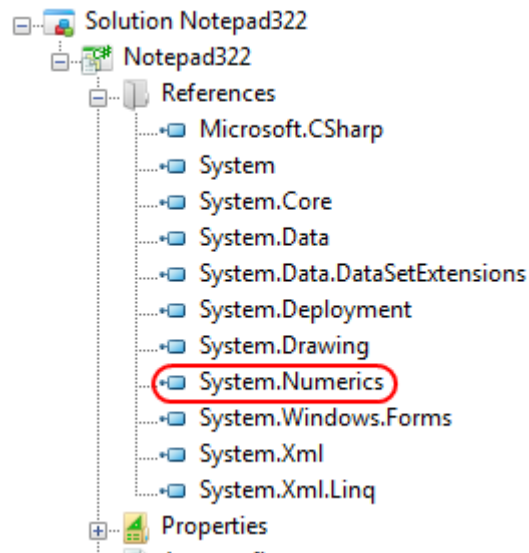
2. Loading from file (1 point)

Add the option to load text from a file. You can use a stream reader to open the file and pass it to your loading function. Remember that [StreamReader](#) inherits from TextReader and has a [constructor](#) that takes a file name as a parameter.

3. Loading from FibonacciTextReader (6 points)

Write a class named FibonacciTextReader that inherits from the System.IO.TextReader. Since you've implemented a loading function in part 1 that takes a text reader, you'll be able to pass this object to your loading function after creating an instance of it. This class must:

- a. Inherit from the TextReader class
- b. Make it have a constructor that takes an integer as a parameter indicating the maximum number of lines available (the maximum number of numbers in the sequence that you can generate).
- c. Override the ReadLine method which delivers the next number (as a string) in the Fibonacci sequence. You need logic in this function to handle the first two numbers as special cases. You must return null after the nth call, where n is the integer value passed to the constructor.
- d. Override the ReadToEnd method if you used it back in the implementation for part 1. Implement it such that it repeatedly calls ReadLine and concatenates all the lines together. You can use the [System.Text.StringBuilder](#) class to append the lines together into one string.
- e. Use the [System.Numerics.BigInteger](#) struct for the sequence numbers. The numbers get beyond what a 64-bit integer can store very quickly, so you need this to accurately compute the first 100 numbers in the sequence. You may have to add a reference to the System.Numerics DLL in your project for this. Right-click references -> Add reference -> find System.Numerics.dll.



- f. Optional: put the line number before the sequence number as shown in the sample app screenshot.
4. Add three menu options (or buttons if you want) in your interface to allow for demonstration of the above functionality.
 - a. There needs to be an option to load from a file. This should prompt the user for a file name (use [OpenFileDialog](#) for convenience) and then load it as described in #2 above.
 - b. There needs to be an option to load the first 50 numbers of the Fibonacci sequence
 - c. There needs to be an option to load the first 100 numbers of the Fibonacci sequence
 - d. (note: parts b and c should have almost identical code, just with a different number passed to the FibonacciTextReader constructor)
5. Put your name and ID number in a comment and the top of each code file that you write. Put a reasonable number of comments in your code.

Point breakdown:

- This assignment is worth 10 points
- 1 is for correct saving functionality
- 8 are for correct loading function
- 1 is for the remainder of items, such as adequate comments in the code, proper interface design, and so on.