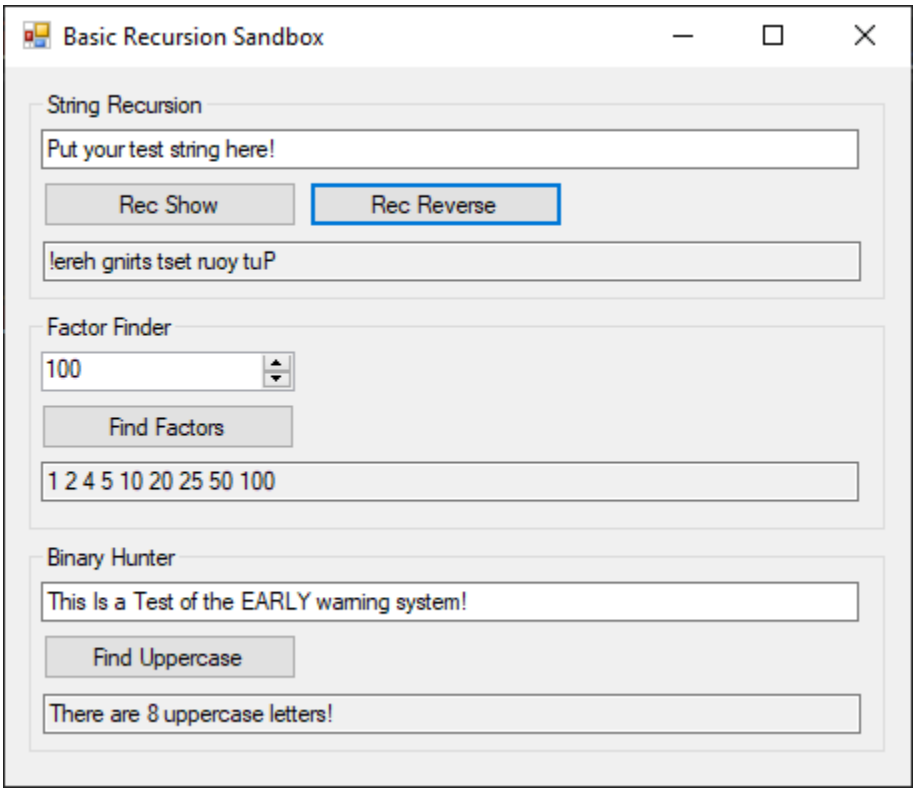
School of Applied Sciences And Technology
Department of IST
Program: CNT

ICA #09 – Basic forms of Recursion

Create a Windows Forms application with the following appearance:



This ICA is consists of four major parts, and you may complete them independent of the other parts.

Part A

The 'String Recursion' section will display a string forwards and backwards using an input textbox, a readonly output textbox, and two buttons. The buttons will clear the output and start the recursive function that you will implement as a separate helper method that is separate from the button event handler.

The 'Rec Show' button will start the call to a recursive method that will accept a character array representing the contents of the input string, a starting index, and an ending index. The index values will

be fully inclusive of the indices for the character array. The function will check to ensure that the starting index and ending index are not inverted, returning if they are. The output textbox will be updated to include itself and the character at the start index, using string interpolation. The function will then call itself with the same values, except the start index will be offset by 1. Because the processing (output) is happening prior to the recursive call, this is a form of *head recursion*. The string should come out forwards.

The 'Rec Reverse' button will operate almost identically to the code above. When implementing the recursive function, simply reverse the order of the recursive call and the text update lines. Because the recursion is now happening after the processing (display), this is a form of *tail recursion*. The string should come out backwards.

Part B

The part will use a numeric up/down control to permit the user to enter a value from 1 to 1000000. Pushing the 'Find Factors' button will start a recursive call to a method that takes a value and position argument; both as integers. The recursive method will ensure that the position is not greater than the value, returning if it is. If the value is evenly divisible by the position, you have found a factor. Update the output text to be itself plus a space plus the position value. Trim the result so there is no leading space. The recursive function should now call itself with the same value and position + 1.

This is an example of a recursive implementation that offers very little over the iterative version. It does, however, offer another opportunity to see how a recursive method is constructed.

Part C

Problems that can be broken into smaller instances of themselves are typically great candidates to solve with recursion. A binary search, for example is a perfect example of this. We can use a very similar approach in real life to break a task out among multiple people.

The purpose of 'Binary Hunter' section is to see how a problem can be broken into smaller chunks for processing. The smaller chunks require the same methodology to solve; they are just smaller.

Unlike the other recursive functions above, we will use the return value from this recursive method to count something we are looking for (uppercase characters in a string). This is sometimes referred to as a 'rollup' value. Error conditions will typically return 0, as you didn't find what you are looking for if an error occurred. If you find what you are looking for, you would usually return this as 1. You need to include the rollup values from any recursive arms that you include.

The 'Find Uppercase' button will start a recursive method call with a string, a low index and a high index. You will determine the error conditions this time, and return 0 from the method if you encounter any.

This function will split the string in half (not literally, manipulation of the index values will achieve this by determining the range of characters we look at). Each half will be recursively pursued, so there will be a recursive call to process the lower half and the upper half of the string separately. Eventually, if we keep cutting the string into halves, we'll run out of string to chop up, meaning eventually the lower and upper index values will be the same.

If the lower and upper index values are the same, check to see if the character at that position in the string is uppercase. If it is, return 1 (you just counted 1 uppercase character). If it is not, return zero. At this point, this end of the recursive arm is complete. This section of code will come immediately after the error checking code.

After the above processing (index values are not the same), find a partition index that is in the middle of the current index values. Recursively call the method with the lower index to partition index, and recursively call the method with the partition index (plus 1) to the upper index. This will cause the processing to continue with the two string halves. Eventually these recursive arms will complete, rolling up the number of the uppercase characters found. To make this work, the method needs to return the sum of the return values of these calls.

Work on an example of this on paper before you begin to code!

Item	Marks	Penalties
UI Design • **UI as shown**	6	Inappropriate component names: -3
Code Design and Implementation • RecShow using head-recursion properly • RecReverse using Tail Recursion properly • Factor Finder finding factors properly using recursion • BinaryHunter counting uppercase letters properly using recursion	4 4 8 8	
Documentation: • Programmer Block • Well commented code • Appropriate Variable Names • Proper spacing between blocks of code		Missing components of documentation: -1 to -6