Lab 2 – Nathaniel Foster

Once again, lab 2 passes a stack structure to the algorithm and then there are two outputs. The first is an output stack passed as an argument which should only have 1 item on the stack representing the new converted equation. The second item is a boolean indicating whether or not the conversion was successful. It returns true if successful or false if the conversion failed for some reason. However, it does not specify the reason for failure.

A stack was a clear choice for use as the output variable as it could be modified and the modifications would be retained after leaving the algorithm unlike a String object the allocates new space in memory every time a new String is defined. A stack was also desirable for receiving the input equation as it would naturally reverse the order of the of prefix equation read from the input file for processing. Additionally, it was desirable to design the function with a similar/identical ADT to the iterative approach with only implementation changes for ease of use. You'll notice, that the only change to the main driver is to now call the recursive converter instead of the iterative converter. The iterative was left in the code but commented out for reference.

The stacks were implemented as arrays again. This was found to generally be a good use of space as the equations can be safely assumed to be a certain size since they are somewhat physically limited by the amount of characters that can fit across a page. It avoids the need to use nodes with an extra pointer to the next node. However, there are frequently many empty cells in the array so some space is wasted. Another issue that had to be dealt with was processing an equation larger than the arrays original size. The Stack class I created offered a constructor to that took the Stack length as an input if it was known so space could be used most efficiently. Otherwise, it defaulted to size 30 and doubled whenever the stack grew to the maximum size that the array could hold. This could potentially lead to a large number of empty indexes in the array which in other applications could be an issue. Because of the large human input to the equation conversion algorithm, this luckily isn't a problem. The last user defined input tests the case with a long invalid equation. However if for example an algorithm is converting data incoming from a network, this method for dealing with stack overflow may not be reasonable. A better design may perhaps be to only grow the array by a percentage so that it will grow at a rate relative to the amount of dating being received. However, this operation would be come increasingly more cumbersome as the entirety of the original array must be copied to the new array. A linked list may significantly less complex operation as the size of inputs grows. This overflow issue is not unique to a recursive or iterative solution.

The recursive algorithm only had one defined stop case which was to stop when the input equation stack was empty. This did lead to some inefficiencies that the iterative solution did not have. If an invalid character was reached or there were not enough operands to process an operator, the algorithm would still continue to pop the input stack and process it but would return false for the success of the conversion. The calling process then had to correctly handle the failure of the conversion. In order to avoid this excess processing, return statements were added to stop the algorithm from making another recursive call. Then each recursive calls success/failure flag was saved to be passed back to the prior call. Additionally, the recursive solution also creates new temp variables every time a new recursive call is made which can lead to more space being used in comparison to the iterative solution. I personally the iterative solution slightly easier read, however prior knowledge of the conversion algorithm made determinate and implementation of the stop case significantly easier. Understanding adding additional stop cases for error conditions was also key to this lab.

As a further enhancement, rather than returning a boolean success value, an integer value was returned as an error code. Again it was up to the function call to handle the error code accordingly. In this implementation, the error code is simply sent to the output. However in other applications, it sometimes makes sense to initiate a reboot of the system as it may indicate a fatal fault that must be cleared via a reboot.