For my postfix_eval program, I began by writing the main function. I started off implementing the easy things such as taking an input via getline and since this was suppose to repeat until an EOF was given, putting it into a while. I then began to write the stack class. My first thought was to have the stack process the information as it was going, resulting in the long-winded constructor. It iterates over each character in order to determine if it's a number that should be added to the stack, or if it is an operator. If it's an operator, it performs the operation on the two most recent entries into the stack. I then originally put errors into the class, making it print to cerr if there were too few numbers present in the stack to perform the operation. This lead to an issue however later on since in order to check if the input was viable, I check if there is only one object in the stack, in which case I print that object. That meant that an input of 4 + would result in an error as well as an answer of 4. That lead to the implementation of the condition and condition2 variables. Condition was implemented in order to ensure there were no abstract symbols and that there would be a proper amount of numbers and symbols for the stack to work properly. Condition2 was later added on when I found out that the unknown symbol was suppose to print a line of characters, not just one character. The section below was added to make sure a chain of "known characters" ex: ++, --, **, etc. were not input. In those cases, condition2 would act both as a way of knowing that multiple known characters were put in, and as a way to know the length of that chain. As long as both conditions were met, it would perform the calculations and when ctrl d was input, the while loop would be exited, resulting in "Bye!" being printed.

For my deque program, I began by looking at the circular array code and implemented most of the same elements, but added some integers "head" and "back" to determine where the head and back of the circular array was. One of the first things I decided however was that the initial capacity was going to be 4. I came to this number after determining what I wanted my shrink percentage to be: 25%. By choosing 4, there is a decent amount of space to begin inputing values while also being easy to calculate 25% from any of its multiples. In addition, since I chose to shrink it by 50%, it makes shrinking extremely viable as it would never result in a fraction. Going through my thought process, I figured it had to be an even number in order to avoid inconsistencies due to truncation during division of ints, and large enough to store a few items while not taking up a ton of space. I also implemented a similar resize program that when used, moved the item placed at head to pos 0 and moved everything accordingly. Everything else went pretty smoothly with the only other thing of note being overloading the [] operator. Originally, I had it printing from the corresponding position before realizing that it was supposed to print relative to head, which was a simple fix after realizing the bug.

For the test deque program, I decided to add 2 more tests on top of the original 3. The first test that I added built off of the first one, checking what would happen if the [] operator was utilized for an object that didn't exist. The second test tested the functionality of the functions Deque<T>::Front() and Deque<T>::Back() and made sure they were properly functioning.