Faustine Yiu
913062973
ECS 36C Program Hw 2 Report

postfix_eval.cc

As the postfix evaluation successfully uses a layer by layer operation to calculate arithmetic, I was able to utilize stack to write a program that evaluates postfix expressions. First, in my input function, I check for spaces, unknown symbols, and return a string of "Bye!" when the command line is exited, aka end of file (EOF) in c++. If the expression entered passes those filters, it becomes the parameter of the function calcPostfix. In calcPostfix, I use sstream to iterate through the expression. If sstream finds a double, I push it to my stack s, but if it sees a char, I need to check if it is an operator via the isOperator function and set my double variable y to the first pushed number in stack s and x to the next pushed number in stack s and pop the numbers I pushed into y and x out of stack s. I then take my operator c, and two numbers x and y as my parameter of the calcOperator function to use the operator c to calculate the two numbers x and y and return a double type result.

Finally, I push this result back into stack s to store as one operation complete. This, as it is in a while loop until all the ss seen expressions are used, repeats itself to use the number originally stored in stack s to calculate with more operations and other numbers. Finally, we will end up with one result to be returned to the command line as our answer. However, if the result holds more than one or is empty, then that means there were some invalid expression that was fed into the program.

luggage_handling.cc:

As the SMF luggage transport simulator, I considered the sentences: "Before departure, luggage is loaded into containers as the luggage from passengers arrives. When a container is full, or there is no more luggage, it is loaded into the airplane…After arriving at destination, the containers are unloaded with the last on being the first off". First, I loaded my stack container, truck, with the bags file data. Then I called my function, calcOutput, to swap the orders of the data in truck enough so that it returns a containers' LIFO order. First, I calculated the remainder of the data size / argv2 to get how many data will not fall into the argv2 size. After receiving that number, I ran a for loop to push only that number of data into the new queue container, called temp. Since our stack truck, is a LIFO container, it is able to return the last of the initial data to temp. Now that temp is a queue container, it is a FIFO, to be pushed onto stack temp2. Now the numbers I need are all backwards, however, I am luck that temp2 is stack, again a LIFO container, and thus it returns the data back to order into our vector output.

In the while loop of the calcOutput function, I check to see if our stack truck has popped all of its data or not, and once it has popped its last set of data, the loop ends. Inside the while loop, I have the same concept of what I had before, but the difference is that this time, it is taking the argv2 input itself, instead of just the divided remainder. To reiterate, I take stack stuck and push its data LIFO to queue temp, then take queue temp and push its data FIFO to stack temp2, then take stack temp2 to push its data LIFO to the final vector output, that already has the first remainder number of data in it. As always, I am able to push the right numbers because I popped whatever numbers I pushed in out after I pushed them in. Finally, I return back to the main function with vector output and print the vector out to the command line. All in all, if there are any errors in input, we return the relating error message.