Sample Program #1 (Smoothie) Testing Strategy

One case was made typed up by hand (small) while the other was made with a program, mklargecase.c. For the small case, I looked up Smoothie King's menu online and grabbed several of the ingredients I saw listed under each of the smoothies. To make sure I was testing somoe bounds, I created an ingredient of size 20, and made sure that wasn't the last ingredient on the list. (The idea here is that often times a missing null character leads to an error in a future string of some sort, so I wanted there to be a future string.) In the small case I created 10 smoothies, one with one ingredient and another with all the ingredients to test the min and max cases of smoothie ingredients. Other than that, I put in my approximations of actual smoothie recipes. Finally I just used 5 stores, one ordering the minimum number (1 smoothie) and another ordering the maximum number (10 smoothies for this case). I made the ratios mostly random. I worked out what the answers to the stores should be by hand.

In attempting to make the maximum case, I realized that doubles in C don't give the required precision to solve the problem. (Doubles at best give precision to a relative error of 10^{-12}, but with 10^5 smoothies at a store with ratios ranging from 1 to 1000 for many ingredients, the build-up of error was simply too much to give answers accurate to 6 decimal places.) **Thus, I created new, smaller maximum bounds for the large test case: I limited # of ingredients of a smoothie to 10, and limited the number of smoothies per store to 34, which was close to what Smoothie King offered.**

To generate the names of items, I wrote a function that converts an integer into a string, essentially interpreting a number in base 26. Thus, the string abwe corresponds to the number $0 \times 26^3 + 1 \times 26^2 + 22 \times 26^1 + 4 \times 26^0 = 1252$, since a = 0, b = 1, w = 22 and e = 4. I just used the names for that correspond to the values 0 to 99,999 for the large test case.

I created a function that generates a random smoothie recipe. The function takes in the number of items for the smoothie and the maximum item number. The hardest part of this function is ensuring that there are no repeated smoothies on the list. I do this with a used array, since the smoothie numbers are limited and indexed from 0. Finally, for each store, I also generate in between 5 and 34 smoothies (from the list of smoothies) to sell. Instead of using the used array, I decided to just write a function that checks to see if a value is already in a list of smoothies. I did it this way since the number of smoothies being generated was pretty small and run-time wise, it seemed like there wouldn't be too many collisions, so the process should go pretty quickly.

There's no way to truly check the answers by hand, but I did write a Java program (where dynamically allocated memory isn't an issue), and both programs agreed with each other, so this is reasonable proof of the answers.