Evan Hedges

CIS 3190: Software for Legacy Systems

Professor Michael Wirth

I underestimated the ease of the assignment. I believed that the due to my familiarity with linked lists, it would be simple. When I first how records were used in Ada, I saw a similarity between records and structures (from C). Since I had knowledge in implementing linked lists in C, I assumed that the Ada implementation would be easy. I began the assignment by creating the user interface, as that would allow me to test different sizes and edges cases while creating my linked list. This was fairly simple and straight forward. Next, I used a source from Rosetta Code to guide my linked list implementation. The actual creation of the list was fairly easy and straightforward. I began by implementing the basic functions, such as insert front, insert back, and display. Once these functions were created, I used numbers of varying sizes to test the linked lists to ensure robustness. The main goal was to prevent linked list errors from appearing when implementing other functions. By doing this, it increases the likely hood that the error was within my fresh code rather than my linked list. Next I began the implementation of addition. This is where the assignment began to get difficult. I broke the addition down into steps. I began by writing out the logic. I immediately realized that adding two numbers with differing sign is subtraction, so that was going to be a call to the subtraction function. Next I looked into breaking down the addition into steps. You would take two values, add them together then check to see if the answer was larger than 10, if so, then you have a carry. Originally, I was uncertain if you there was a larger carry value than 1 (as in somehow when you add, get to a 20.). Thus, my first implementation accounted for large carries. This would be useful later for multiplication, since they can have carries larger than 1. This carry would then be added to the next number. My first implementation of the algorithm resulted in infinite loops and other bugs. After multiple attempts to the fix my problems, I had to scrap the entire function and rewrite it from scratch. I used the same logic as the first attempt, but I was successful in my second attempt. However, after doing multiple additions, I came to the realization that my original set of list instructions were incomplete. I had yet to implement a function to clean the list, thus the numbers had a tendency to be incorrect after a second addition. These errors and others led me to create a testing document with a variety of possible issues. I would use this to track what I tested and investigate possible areas that may contain errors. Thus, the testing document became a guide for the rest of the assignment, allowing me to focus on the basic cases first and slowly adding complexity as the assignment progressed.

Next I began the implementation of subtraction. I began with the basics by checking signs. If they differed, this meant the number would grow, and would use my addition function. Next I reviewed my addition function. My original thought was that I could use my addition function and invert the values to subtract. This did not work, and I deleted the function. I began again, this time by creating using breaking down the steps to subtract and implementing that. I was able to subtract two numbers, however when subtracting a larger number from a smaller, resulting in an inverse sign, the answer was wrong. I went back to the drawing board, and began randomly subtracting a large value from a smaller one. I came to the realization that the value was the same as the positive, however with the sign flipped. This made me realize that I could use my addition function to subtract. In order to subtract, I would need to add the negative value. Armed with this knowledge, I started to implement my revelation. This enlightened me to the fact that I would need to have another list. I created a function to copy the list

into a new one. Once that was complete, I created a loop to run over the list and turn each node's data negative. With this created, I was then able to subtract two values. Next, I began to write code to ensure that the larger unsigned number was on top. I began by comparing lengths, if the second number (subtrahend) was larger, then I knew that I would need to subtract the first value (minuend) from the subtrahend, then flip the sign. I completed this, however the Ada warnings about the operands were annoying. Thus, I added two dummy heads to get rid of them. The final step was to check which value was larger if the lengths were the same. This was fairly simple, iterating over the lists and seeing which value was larger. Then I would either subtract normally, or flip the minuend and the subtrahend. Thus, the third iteration of the function was successful and completed.

Multiplication was fairly straight forward. Although, like my previous functions, it took multiple attempts. My first attempt was confusing and I was unable to debug it. My second attempt was a slow process, following my testing document, I began by creating simple multiplications and adding to it. I started with single value to single, then single to multiple values, then multiple to multiple. I had taken my carry code from addition and modified it to carry values. Ultimately, my biggest issue was when I forgot to pad the right side with zeros. Without them, I was adding ones to ones, instead of ones to 10s and so forth. Overall, multiplication was fairly simple.

Calculating the factorial was annoying. Originally I tried to implement the entire function in one go. However, this did not work out well and I restarted. I proceeded to implement single digit factorials. I wrote this as a separate section because I was unsure how to have the program recognize when a factorial computation was complete. Writing this section was fairly simple and straight forward. Writing the rest of the function was not. During the testing phases, I discovered two bugs; a carry bug in my multiplications and a subtraction bug. The carry bug the carry variable was not reset to 0. This made one digit out of a large string wrong. This complicated implementing factorials because one off digit can lead change the entire number. However, once I began debugging, it was a simple fix. The other subtraction bug was not. What happened was that when I was subtracting, the value was being set to negative then added. However, the value was still negative afterwards. When subtracting again, it added the value back. This created an infinite loop, because I was checking for when the factorial was near completion and had single digits left to calculate. By adding it back, the length never decreased, and infinite looped. My original thought that since the list was an in value, the negative would disappear. This was not the case. Once I discovered this, the fix was simple. Then the assignment was done.

Overall, the assignment was not an easy one. I underestimated how to do simple operations such as multiply or subtract. I had to often rewrite entire functions or take a break before coming back to it. Sometimes the output would be correct, other times it would be completely off, sometimes the bugs were very minute and could be easily missed. This was frustrating because I was running into problem after problem in a project that I assumed to be simple. Often times I would work on the function for the entire day, have numerous issues, then take a break. After the break I would delete the entire function and rewrite in a few hours and fix it. The end result was that the past hours felt wasted. I had spent the day trying to accomplish something, then end of completely rewriting and fixing it in two to three hours. However, it did make me realize that I should slow down before coding. Since, I thought that the assignment was easy, I did not need to plan as much. However, my experience showed that I should take more care in planning and focus more on design rather than implementation.