CSE2225 – DATA STRUCTURES PROJECT 1

Multiplication of Two Linked Lists that can have infinite digits

Beyza Nur Kaya - 150120077

CSE2225 Data Structures, Fall 2022-2023

Date Submitted: Nov. 18, 2022

IMPLEMENTATION

In this project, our topic was to calculate the multiplications of numbers between base 2 and base 10 using a linked list. This helps us to calculate the multiplication of numbers that are higher than the intmax, the maximum number a computer can represent.

My way of solving this problem was creating a doubly linked list. Since the mathematics of multiplication is based on the multiplier and multiplicand's reverse, that should have been helpful in many cases. Then, I wrote a function that calculates the product and at the end I've made a function to convert the bases.

To give details, I created a NumberNode with an int number as data, also since it is a doubly linked list, it also has previous(prev) and next pointers. Then, I created two functions so that I can insert numbers into the linked list. One inserts from the tail when the head is constant. Another one inserts from the head when the tail is constant.

Then I created 2 functions to print the linked list. One prints from head to tail; other one prints from tail to head. But since some linked lists have 0 at their beginning, I needed to delete them while printing the reverse. I made a count and if the number that tail has is not zero and it is the first time finding a nonzero number, I increase the counter by 1 and while counter is 1, we print the linked list.

Then I have the multiplyLists function. First, I've tried to hold each carry, but it was so prone to errors, so I decided to put the carry directly into the list without losing any carries while looping and summing the subproducts. First, not to have a segmentation fault, I've put 0 into first element of the result. Then, I made 4 pointers to the multiplicand's tail, multiplier's tail and result's tail and head. I assigned a carry (which will have all carries at once unlike my first attempts) as zero since the first multiplication does not have a carry mathematically. Then, I looped through the current multiplier which points to tail of the multiplier, inside of this loop I looped through the current multiplicand which points to the tail of the multiplicand. Then, I've created 2 more pointers to the result pointer – so that I won't be losing my result's head but also can add the carry to the right position. I made a total variable which has the summation of the current multiplication multiplied by the current multiplicand and the carry (at first zero, then assigned according to the total, also carry is the number of the next of the currentResultHeadForLoop so I can also use that). Then, I created a digit variable, that holds the digit which should be in the linked list, and it is calculated as total modulo base. Then I assigned carry as the integer division of total and base. Then I said the position of this digit should be the head of the current result's number by using the currentResultHeadForLoop, which is basically position pointer. Then, since we are given result as NULL, generally the currentResultHeadForLoop's next will be NULL. Because of that, I've added 0 to the base result linked list which is also in arguments of the function. Then, I carry the carry to the next digit of the result with adding into the old values not to lose any carry again. Then I changed the position of currentResultHeadForLoop to its next and current multiplicand (multiplicand's tail pointer) to its prev. Then, the inner loop is done. We return to the outer loop and make result's head pointer to it's next so that we can shift the carry addition by one each time a multiplier is done with multiplying with multiplicand. I make carry 0 again, so that it won't have any values inside. Then I changed the current multiplier (multiplier tail pointer) to its prev. Also, make the current multiplicand start with the tail of the multiplicand too. This will loop until there's no multiplier digit left to be multiplied.

Then I have the baseMultiply function. I've created a simpler version of multiplyList but this one only multiplies pairs where multiplier is one digit. Since base conversion takes a lot of time, to reduce time I've done that – less if conditions and directly gives the result, not the reversed result. Basically, this function loops from every digit of multiplier and inside of this loop, it also loops from the multiplicand's every digit. When multiplicand is finished, goes to another multiplier, and starts again looping from multiplicand. In the inner loop, we calculate the product of the digits we have and add carry into that. After finding the total, its modulo base is the digit which should be left in the linked list, and its integer division by 10 will be the new carry and so on. We insert the digits as we found them into the result linked list. Eventually, the result linked list has the multiplication the multiplier and multiplicand with a non-reversed order.

Then I have a function that is called addTwoLists. This function adds two linked lists. We loop through both linked lists and then add them into an added variable with the carry. According to their being null or not, we shape added using conditionals. After calculating added, its modulo base is the digit which should be left in the linked list and its integer division by 10 will be the new carry and so on. We insert the digits to the result linked list and continue looping from the tail to the head of each number which will be added. Finally, we will get a result linked list that contains the sum of these two linked lists.

Lastly, I have a function that is called baseConversion. Here I had to do some declarations and initializations first. Then I form 4 linked list, 3 of them are inserted just base, 1 of them is inserted just 1. Then I create a count variable to count the exponent is whether 0, 1 or bigger than 1. If it is 0, then the exponent should be 1. If it is 1, then the exponent should be simply base. If it is bigger than 1, we need to multiply them until the end of the conversion. (i.e., for base 6, 1 ,6 and 6*6, 36*6, 216*6...) Then, when we start to convert them. In conversion, we do this calculation: digit * base ^position of the number and we add these all while we loop through to the end position. Since I've calculated the bases exponents, I simply multiply the bases exponent to the corresponding digit using multiplyLists. Then I add these (the new founded and old summation) to a result linked list while looping. After finding it, I print them reversely since they are reversed – because of the multiplication. Then if the count is less than 2, I increase the count, so that if the exponent is bigger than intmax, I could have count as 2. Also, I made this function capable of doing 3 conversions at a time so that we won't lose the base's exponents and redo it again. If I call it three times, that means the time it takes would be so much bigger than doing it at once.

In the main function, I take the file and assign them to a linked list. Since we will execute the program using arguments, I open the file if the argc is equal to 2. If it is larger than 2, I give a message like try writing one argument – one input file name. Then I create the node pointers for multiplicand, multiplier, and its product. Then, I initialize base and count as 0. The count is increased by 1 each time it the char we get is new line. Then I read the input file from char to char and assign them to the linked lists by inserting. After that, the last line is base, we assign the base as well. Later, we close the file since our job with it is finished. Then, we simply call functions to print multiplicand and multiplier, to multiply multiplicand and multiplier, to print result reversely, then converting the bases too. That's the end of the whole program.