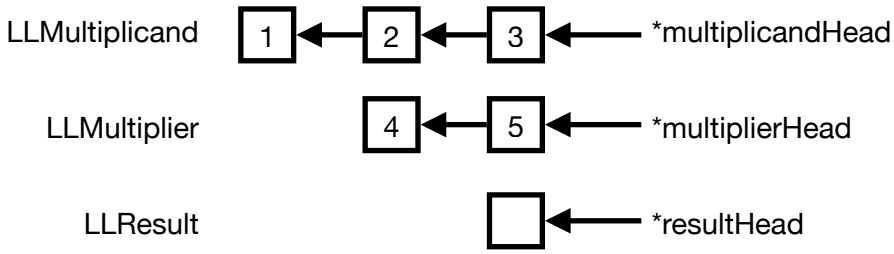
Berkay Mengunogul 150119934

CSE2025 Data Structures PROJECT #1

Report

In this report, first I will explain my algorithm with illustrations then I will present how I implemented it. At last, I will show how to use it and I will share the results.

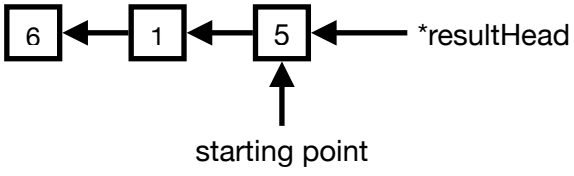
First, I created three linked lists which are representing multiplicand, multiplier numbers and one more for keeping the result. I get the multiplicand and multiplier from a text file. Let's assume that we have "123" as multiplicand and "45' as multiplier. So, linked list that keeps the result is empty for now. How it looks is shown below.



Then every element of LLMultiplier will be multiplied with every element of LLMultiplicand one by one. Through the process of each multiplication, result is separated in 2 parts such as "carry" and "value". Variable carry will hold the tenths digit of the result and it will be added to the result on next iteration, variable value will hold the ones digit of the result and it will swapped with the iterated element of result. Iterations are shown below.

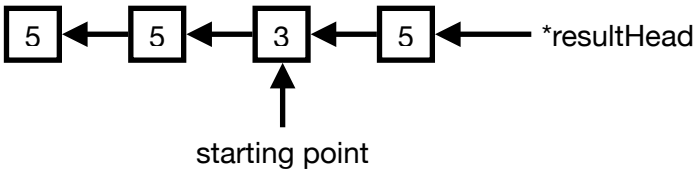
Iteration	Multiplier	Multiplicand	LLResult Current Node	Result	Carry	Value
1.1	5	3	EMPTY	15	1	5
1.2	5	2	EMPTY	11	1	1
1.3	5	1	EMPTY	6	0	6

So, at the end of iteration 1.X LLResult should be looking like shown below.



Iteration	Multiplier	Multiplicand	LLResult Current Node	Result	Carry	Value
2.1	4	3	1	13	1	3
2.2	4	2	6	15	1	5
2.3	4	1	EMPTY	5	0	5

So, at the end of iteration 2.X LLResult should be looking like shown below.



Second, I am going to explain how I implemented this algorithm in C. I implemented a structure called node for each element of linked lists which int value and a pointer for next element.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Node structure for linked list
typedef struct node{
    int value;
    struct node* next;
} node;

// Function for inserting a node to a given linked list
void insertNode(node **head, int value){
    struct node* newNode = (node*) malloc(sizeof(node));
    newNode->value  = value;
    // if starting pointer doesnt indicate any adress first node will be directed to the starting pointer
    if(*head == NULL){
            *head = newNode;
            return;
        }

    // It skips over till last node where next node is empty
    node *currentPtr = *head;
    while (currentPtr->next != NULL){
            currentPtr = currentPtr->next;
        }
    // links the new node to the end of linked list
    currentPtr -> next = newNode;
}
```

To insert elements to linked lists, i wrote a insertNode function which uses a header pointer of linked list and value for the new element. If the given linked list is empty then header pointer indicates nowhere, in this case first node created will be indicated by the header pointer. Otherwise, with while loop, a pointer is scanned through the linked list till the last element. After accessing the last element new created node added to the end of linked list.

```c
// Function for multiplying the linked list and saving it into a another linked list
void multiplyLists(node **multiplicand, node **multiplier, node **result){

    // multiplier digits
    node *currentMultiplierPtr = *multiplier;
    int multiplierIndex = 0; // to track the index of multiplier di
    int carry = 0; // to hold the carry digit of multiplication for next iteration
    while (currentMultiplierPtr != NULL){
            // multiplicand digits
            node *currentMultiplicandPtr = *multiplicand;
            node* currentResultPtr = *result;

            // it skips the multiplier digits by the index of multipler
            // The reason is every currentMultiplierPtr digit is multiples of 10**index.

            // to illustrate;
            //   12
            //   23
            //x____
            //  36
            // 24  --> 6 is skipped
            //_____
            // 276

            for(int i = 0; i <multiplierIndex; i++){
                    currentResultPtr = currentResultPtr->next;
                }

            //goes through multiplicand digits till the last digit/
            while (currentMultiplicandPtr != NULL){
                // if linked list of result indicates empty node then it inserts
                if (currentResultPtr == NULL){
                    int value = currentMultiplierPtr->value * currentMultiplicandPtr->value + carry;
                    carry = (value / 10)%10; // update the carry in here
                    insertNode(result, value%10);
                    currentResultPtr = *result;
                    while (currentResultPtr != NULL){
                        currentResultPtr = currentResultPtr->next;
                    }
                // else it updates the value of current node
                } else {
                    int value = currentMultiplierPtr->value * currentMultiplicandPtr->value + carry + currentResultPtr->value;
                    carry = (value / 10)%10; // update the carry in here
                    currentResultPtr->value = value%10;
                    currentResultPtr = currentResultPtr->next;
                }
                currentMultiplicandPtr = currentMultiplicandPtr->next;
            }
            // at the end of the multiplication if there is any carry left, it will be inserted to linked list
            if(carry != 0){
            insertNode(result, carry);
            carry = 0;
        }
            // update the pointer to next multiplier digit
            currentMultiplierPtr = currentMultiplierPtr->next;
            // update the value of multiplier digit
            multiplierIndex += 1;
        }
        // at the end of the multiplication if there is any carry left, it will be inserted to linked list
        if(carry != 0){
            insertNode(result, carry);
            carry = 0;
        }

    return;
}
```

When it comes to the multiplication function, first it iterates through multiplier digits within iterating multiplicand digits. Index of multiplier digit is counted at int multiplierIndex variable. It will be used for deciding the starting point of LLResult. With regards to index of multiplier digit, starting point of LLResult will be changed. With currentResultPtr pointer, order of the processed element tracked for LLResult. It changes to next element every multiplication. During the multiplications, digits multiplied, carry and value of current LLResult element added to each other. Tens digit of the result kept at carry and ones digit of the result swapped with the current value of the LLResult element. int carry variable is 0 at the beginning and it is updated with each multiplication then it is added to next multiplication. Before the multiplication, if the currentResultPtr indicates nothing then it inserts the result as a new element at the end of LLResult. Otherwise, value of the indicated element will be updated. Before going to next multiplier, if there is a care which is not 0, it will be inserted to the end of LLResult.

```c
// Fuction for reversing a linked. It is need beacuse while multiplying the numbers
// process happening from right to left
// when it comes to printing the result it happens from left to right.
// Since one-way linked lists used in this project it is a necessary function
void reverseList(node** head){
    node* prevPtr = NULL;
    node* currentPtr = *head;
    node* nextPtr = NULL;
    while(currentPtr != NULL){
        nextPtr = currentPtr->next;
        currentPtr->next = prevPtr;
        prevPtr = currentPtr;
        currentPtr = nextPtr;
    }
    *head = prevPtr;
}
```

reverseList function is simply reverses the linked lists. The reason why I implemented this function is while reading inputs they have been read from left to right. But multiplication is happening by reading digits from the right to left. Therefore, I had to reverse the inputs before multiplying them.

```c
// write the results to the ouput file
void printListsToFile(node **multiplicand, node **multiplier, node **result){
    FILE *outputFilePtr;

    // With using reverseList function, before printing the results I reversed the linked lists
    // one by one
    // The reason is digits are multiplied right from left but printed the opposite way.
    reverseList(multiplicand);
    reverseList(multiplier);
    reverseList(result);

    if ((outputFilePtr = fopen("output.txt", "w")) !=NULL) {
        //write multiplicand
        node *currentPtr = *multiplicand;
        while (currentPtr != NULL){
               fprintf(outputFilePtr, "%d", currentPtr->value);
                currentPtr = currentPtr->next;

           }
        fprintf(outputFilePtr, "\n");
        //write multiplier
        currentPtr = *multiplier;
        while (currentPtr != NULL){
               fprintf(outputFilePtr, "%d", currentPtr->value);
                currentPtr = currentPtr->next;

           }
        fprintf(outputFilePtr, "\n");
        //write result
        currentPtr = *result;
        while (currentPtr != NULL){
               fprintf(outputFilePtr, "%d", currentPtr->value);
                currentPtr = currentPtr->next;

           }
    }else{
        printf("File could not saved");
    }
}
```

printListsToFile function is reverses the inputs back to normal and one by one reading the elements of LLMultiplicand, LLMultiplier, LLResult, it writes the results to a output.txt file.

```c
int main(int argc, char* argv[]){
    FILE *inputFilePtr;
    node* multiplicandHead = NULL;
    node* multiplierHead = NULL;
    node* resultHead = NULL;

    // main function gets the dir of input file from command line
    // main function gets the dir of input file from command line
    if ((inputFilePtr = fopen(argv[1], "r")) != NULL) {
        char input_multiplicand, input_multiplier;

        // creates a linked list for multiplicand from input
        while (((input_multiplicand = fgetc(inputFilePtr)) != EOF) && (input_multiplicand != '\n'))
        {
            insertNode(&multiplicandHead, atoi(&input_multiplicand));
        }

        // creates a linked list for multiplier from input
        while (((input_multiplier = fgetc(inputFilePtr)) != EOF) && (input_multiplier != '\n'))
        {
            insertNode(&multiplierHead, atoi(&input_multiplier));
        }

        //reversing the linked lists beacuse numbers are multiplied right to left
        reverseList(&multiplicandHead);
        reverseList(&multiplierHead);

        // multiples
        multiplyLists(&multiplicandHead, &multiplierHead, &resultHead);
        // puts the result to the output file
        printListsToFile(&multiplicandHead, &multiplierHead, &resultHead);
    }
}
```

In the main function, I have created node type pointers as header pointers for linked lists. Then, input file is read. While reading the input file, linked lists for multiplicand and multiplier created. After that, they are multiplied with multiplyLists function and results are written into output.txt with printListsToFile function.

```
OUTPUT   TERMINAL   DEBUG CONSOLE

project_1 % gcc berkaymengunogul150119934.c -o berkaymengunogul150119934
project_1 %
```

← Compiling the program

```
OUTPUT   TERMINAL   DEBUG CONSOLE

project_1 % gcc berkaymengunogul150119934.c -o berkaymengunogul150119934
project_1 % ./berkaymengunogul150119934 files/input.txt
project_1 %
```

← Calling the program with directory of input file

Finally, after compiling the program, program should be called with the directory of the input file.

project_1 > files > input.txt

```
1  564756754656854675664567456745674567
2  6786757685678
```

← Example input file

Input file should be a txt file with inputs are given at separated lines.

project_1 > output.txt

```
1  564756754656854675664567456745674567
2  6786757685678
3  383286724520597308765203313937018869686937698647514 26
```

← Example output file

Output file should be appear at current working directory and result is the 3rd line as shown above.