CSE225 Data Structures PROJECT #1

Table of Contents

[Purpose of the Program 1](#_Toc180696767)

[Input File Format 2](#_Toc180696768)

[Functions 2](#_Toc180696769)

[outputF(); 2](#_Toc180696770)

[insert(); 2](#_Toc180696771)

[insertR(); 2](#_Toc180696772)

[multi(); and orderPtr() 3](#_Toc180696773)

[placePP(); 4](#_Toc180696774)

[printList(); 4](#_Toc180696775)

[Explanation of Operative Logic of the Program 4](#_Toc180696776)

[1 By 1 Multiplication 5](#_Toc180696777)

[Karatsuba Method 6](#_Toc180696778)

[Reference 7](#_Toc180696779)

[Notes 7](#_Toc180696780)

# Purpose of the Program

This program is created to multiply two numbers that has theoretically infinitely many digits

Program uses a linked list implementation to handle the numbers. That gives us flexibility to use as many digits as we want. It uses a distinct multiplying function for computing the result.

The program also has file read and file writing implementation.

# Input File Format

Input file should have the following format:

multiplicand: number\_of\_your\_choise

multiplier: number\_of\_your\_choise

Places of white spaces do not affect the input but places of multiplicand and multiplier (the strings) should not be interchanged and the ‘:’ characters are necessary.

# Functions

## outputF();

This function exists to handle output files. Function starts by creating a file pointer for output file and opening the file in writing mode. After that comes to make sure that there will be no mistake writing the numbers. Since the numbers - multiplicand, multiplier and the result- uses linked list implementation it is possible for their headPointer to get miss located. Function makes sure that their headPointer points to the right position and since my linked lists work as reversed double linked list for correct output it travers to the end of the linked list which is start of the number. It also prints the execution time of the program which is sent as a parameter of double type to the function using time.h library and clock() function.

## insert();

insert() function is a typical insert function where you can see every program that has to do with linked list . The only difference in function from typical handling is that it adds the nodes to the linked list in reverse order that is first node added to the list is always the tail node and the last node added to the list is always the head node.

## insertR();

insertR() function is nearly the same as insert function the only difference is that it takes one more pointer to pointer as parameter to define a pointer that will point to the tail of the list that is the first node added to the list. – remember that linked list is a reversed double link list- It is implemented to make easier and faster some of the computation and also file writing.

## multi(); and orderPtr()

multi() function is where all of the calculations are done. Basic form of the iterative calculations is the form:

This equation is the basic of calculating what will be the next node in the result. As you can see addition and multiplication done on the same node which will reduce the exaction time compared to creating multiple nodes for every addition operation we are going to need. (For example, to make a 2 by 2 multiplication by hand first we find the multiplication of first digit of the multiplier and the multiplicand and write it to the result after skipping one digit at the result section we do the same thing for the second digit of the multiplier and then sum them.) Having said that this implementation has problems in the first iteration since in the first iteration there are no nodes in the result where we can change it is num field nor do we need to consider sumRem. Therefore, to solve that problem this equation becomes available after the first iteration.

In the first iteration of the program only the first digit of the multiplier multiplied by all of the digits of multiplicand one by one and inserted to result. It uses basic multiplication equation:

There is orderPtr() used to mimic the digit skip step when we do addition operation. Since the first equation updates the num variable of a result node changing the pointer that points to the node where the operations will take place, by one node to the left mimicking the digit skip.

## placePP();

This function inserts decimal point to the correct position in the result linked list. It does not do any calculation, and it is only purpose serves for the output file.

## printList();

printList() only used for making debugging easier and does not serve any other purpose but left in to show the work. It prints the node in a manner that no typical printList() method will do. It helps to be sure that placePP and orderPtr functions work and calculation are done correctly.

# Explanation of Operative Logic of the Program

The most basic multiplication operation that you can do is probably multiplying two integers with one digit. And that is what we do when we are calculating the multiple of two numbers by hand. When it comes to multiplying numbers that has decimal it does not change the process but adds a step where we sum how many digits are there after the decimal for both numbers multiplier and multiplicand and sum them to find how many digits there are going to be in the result after the decimal point.

Computers uses different multiplication logic since system works on a 2-basis system. Since there are restrictions for the variables that you can use such as INT\_MAX being 2147483647 it is not possible to make a multiplication that makes the result more than 10 digits since it is not possible to express the integer result in that increased interval. It is possible to bypass that restriction with other variables such as long long and double which can store bigger values and can represent the numbers in decimal as well. However, that does not change the fact that there is still constriction but what we want is to theoretically being able to multiply two numbers that have infinite digits in which the result is also going to be infinitely many digits.

Only way to achieve this is to use structures and create a linked list to store numbers that we are going to use. If you think about it that does not solve the problem of having infinitely many digits since we have memory constrictions, but it let’s us to go really high digits as long as our memory allows. My linked list stores the numbers’ digits on the linked list as a reversed double linked list where the head of the linked list hold the last digit of the number and also the node that lastly added to the linked list whereas the first node is the last node. Reverse store is not necessary, but it is convenient because if you think about it we start multiplication from the last digit when doing it manually. (Figure 1)

A black screen with numbers and arrows

Description automatically generated

Figure

## 1 By 1 Multiplication

Creating a linked list for solving the problem of constrictions of predefined data types is one thing the other thing is that we have to create a new multiplying algorithm that uses that linked list to make everything work. I took a direct approach there and implemented the actions we do when we work on multiplication by hand. This method is actually allowing us to make multiplication of two infinite digits possible.

A black screen with writing on it

Description automatically generated

Figure 1.2

(Figure 1.2) Thinking about how we do multiplication of numbers that has several digits we basically take the one of the numbers’ single digits and iterate through other one’s digit while multiplying until end of the number that we choose to iterate, taking remainders into account. Following of the process we take the next digit on the number that we firstly used and then again iteration through the digits of the latter one. We write the results of that multiplications, skipping one digit to the left each time and then summing them at the end. My method of multiplication algorithm uses this conventional method to multiply two numbers that are stored in the linked lists and writes the result into another list.

Multiplying decimal numbers are the same there is no difference except we have to count each numbers’ digits after decimal points and add a decimal point to the result number where there will be sum of the multiplicand’s and multiplier’s number of digits after the decimal point in the result. I do this counting process while file reading where I store the number of digits in another linked list since it can be theoretically infinite as well therefore it wouldn’t solve the problem if we use predefined data types as counter.

Considering that both multiplier and multiplicand have n digits our time complexity can be showed as (big oh notation). There are clearly faster methods for multiplying and also multiplying variables that computer does is far more faster than any other algorithms we can think of none of them lets us multiply infinitely many digits (none I could think of at least) .

## Karatsuba Method

The Karatsuba algorithm is a fast multiplication algorithm. It was discovered by Anatoly Karatsuba in 1960 and published in 1962. It is a divide-and-conquer algorithm that reduces the multiplication of two n-digit numbers to three multiplications of n/2-digit numbers and, by repeating this reduction, to at most single-digit multiplications. It is therefore asymptotically faster than the traditional algorithm, which performs single-digit products.

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

It is clear that Karatsuba method makes computing faster but when it comes to our problem having infinitely many digits means that we can have numbers bigger than 2.5 times of INT\_MAX which makes it impossible to divide the number into two segments using only two variables so we need to divide the number more than 2 sections which will increase the execution time – big oh is still but practical execution time will increase- and also might cause problems like stack overflow since recursive function calls will be to much to handle. I didn’t implement this method because I didn’t think it will make that much of a difference and I wanted to have my original method. However, it is reasonable to use Karatsuba method in the program or at least while doing some of the multiplications that has numbers with lower digits. Also I think that this method defiles the purpose of the program since we wouldn’t need any linked list or structure and also it works like where you will force program to use unsigned long long for variables and keep numbers store at that variables that has much bigger storage value, at the limit you create another one to make calculations only using this variables; which makes implementation easier and program faster but as I said defiles the purpose. I wanted to talk about it just because it affects execution time greatly and proposes a new perspective to the problem.

# Reference

<https://en.wikipedia.org/wiki/Karatsuba_algorithm>

# Notes

I compile the code with -std=c89 flag ( ansi c standard) and -pedantic (forces c89 ).

Program is tested on WSL Ubuntu sub environment for windows.

Info about the executable file: file 150123060

150123060: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=47821e39286f1bfc2a6187143842ffaa96202fce, for GNU/Linux 3.2.0, not stripped