Design and Analysis of Algorithms (UE15CS251)
Assignment-1 Report

Name: Aviral Joshi
USN: 01FB15ECS062

Problem Statement:

Design and implement a C library for integers of arbitrary length (“intal” in short for integers of arbitrary length). It should have functions to read and print “intal” and mathematical operations on “intal”. The integer could be positive, negative or zero. The following functions to be implemented on “intal”.

(a) Add two integers of arbitrary length
(b) Subtract two integers of arbitrary length
(c) Multiply two integers of arbitrary length
(d) Division limited to integer division
(e) Exponentiation limited to positive power.

Write a demo program to demonstrate the functionalities of the library.

Files:

intal.h – header file includes function prototype, structure defination and 2 statically allocated temp. char arrays

intal.c – contains All the functions used in the program

intal_demo.c - demonstration of the code

Approach, Program Summary:

Structure “intal” members:

num -> char pointer : points to the dyanmically allocated char array
length -> long int : holds the size of the num array
sign -> char: holds the sign of the number (‘+’ , ‘-’ , ‘0’)

Addition:

intal ***add_intal**(intal *, intal *);
calls one of the following :
-> intal *add(intal *, intal *);
-> intal *sub(intal *, intal *);

Addition function **add_intal** deicides whether to add or subtract based on **sign** of the given intal variables calls add implicatly calls **add** or **sub** function as needed

Subtraction:

intal ***sub_intal**(intal *, intal *);

calls one of the following :

-> intal ***add**(intal *, intal *);

-> intal ***sub**(intal *, intal *);

Subtraction function **sub_intal** deicides whether to add or subtract based on **sign** of the given intal variables calls add implicatly calls **add** or **sub** function as needed

Multiplication:

intal ***mul_intal**(intal *, intal *);

calls the karatsuba function :

-> intal ***karatsuba**(intal *, intal *);

calls the **karat** function as well as the **mul** function

-> intal ***mul**(intal *, intal *);

multiplies the intal type variables of size < 3

-> intal ***karat**(intal *, intal *, intal *, long int);

computes $(p1.10^{n} + (p3-p1-p2).10^{m} + p2)$

Multiplication function **mul_intal** calls function **karatsuba** to perform multiplication by using the ***karatsuba algorithm*** thought in class. Which also calls the **karat** function and the **mul** to compute the final answer.

Division:

intal ***div_intal**(intal *, intal *);

calls the function div_rep_sub:

-> intal ***div_rep_sub**(intal *, intal *);

Divison function **div_intal** uses **long division** method to compute the quotient. Each digit of the quotient is computed by the **dev_rep_sub** function which uses repetitive subtraction.

Power:

intal ***pow_intal**(intal *, intal *);

calls the function **mul_intal**

Power function **pow_intal** uses repetitive multiplication to compute the power of a number.

Learning:

Developing an algorithm that handels **carry** for addition and **borrow** for subtraction
Multiplication using **karatsuba** for long numbers grater than long long int type.
Implemented Division using long division instead of the repetitive subtraction approach.

Improvements:

1)

Division was **implemented** using **long divison** method which **performs much faster** in general and **especially** when the **dividend >> divisor**

ex. 1600 / 2 :

-> Number of operation with long division:

= **length of the dividend * number of repitative subtractions per digit in the dividend**

= 4 * (1 + 3 + 1 + 1) = 4 * 6= **24 Subtractions**

-> Number of operations with repetitive subtraction:

= **Dividend / Divisor**

= 1600 / 2 = **800 Subtractions**

Therefore Implementing Long Divison method **Reduces time taken** for the division of intal variables significantly.

2)

Multiplication was implemented using the **karatsuba algorithm** tought in class, Which is **superior** to the **brute force** method and the **repetitive addition** method.

3)

Avoided the use of **memset** on the statically allocated arrays, by carefully using the NULL value i.e 0 at the end of the char arrays to reduce time delay caused by **memset.**

```
char temp_str[100000000];
char temp_str2[100000000];
```