BINK
BIG NUMBER COMPUTATION

By Aman & Sanidhya

Project Description

This project aims to enrich the C programming language by introducing Big Integers support, enabling seamless manipulation of extremely large numbers beyond standard limits. The focus lies on fundamental arithmetic operations like Comparison, Addition, Subtraction, and Multiplication. This enhancement empowers C programmers to tackle complex mathematical challenges, such as Factorial and Fibonacci calculations, with precision and efficiency. By bridging this gap, we aim to elevate C's capabilities and offer new avenues for innovation in numerical computing.

Implementation

The following functions are implemented on “BINK”.

(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.

Approach and Assumptions

- No assumptions were made in case of Addition, Subtraction, Multiplication, and Division.
- The Approach was to store the numbers as character arrays in a structure, and also to store the length of the character array.

Addition

- Brute Force : Addition of two numbers done from right to left, taking carry into account. Call forwarding to subtract function was done when the sign of the two numbers were different (signed addition).

Substraction

- Brute Force : Subtraction of two numbers done from right to left taking borrow into account. Call forwarding to add function was done when sign of the two numbers

Multiplication

- Karatsuba Algorithm : Multiplication of two numbers was done using this divide and conquer algorithm. This method is faster than the traditional brute force algorithm and hence is a better pick to multiply large numbers.
- Problems faced : Due to the recursive nature of this algorithm, there were some problems with the memory allocations and the arrays being returned, which were eventually fixed. Another concern which I faced was the changing of the original parameter after the functions were executed. This was later fixed by making deep copies of the structure variables (parameters), the reason being : copying a structure variable as "a = b" creates only a shallow copy.

Division

- Modified Brute force algorithm : The brute force algorithm for the division of one number by the other consists of repeated subtractions. As the subtraction algorithm was also that of brute force, the time taken to subtract one number by the other n number of times takes longer than expected. Hence, the solution was to modify this algorithm to reduce the number of subtractions by adding multiplications by 10 (which does not take too long as it is just adding trailing zeros). This method worked faster than the traditional brute force division as the subtractions are done in multiples of 10 instead of 1. In this method, we multiply the divisor by the largest power of 10 such that it does not exceed the value of the dividend. We then subtract the divisor from the dividend till the dividend becomes smaller (this will be value used in the next iteration, let us call this "val") than the divisor and keep track of the number of subtractions. This "count" will be the first digit of the quotient and hence, we append this to the result being returned. The process is then repeated for "val" which will be the dividend and the divisor would be divisor/10.
- For example : 3000 / 1 would require 3000 subtractions according to the traditional brute force algorithm. But, with the above mentioned algorithm, we will need 3 multiplications by 10 and 3 subtractions.

Exponentiation

- Assumption : The Exponent is positive.
- Brute Force Algorithm : The number was multiplied by itself “exp” number of times when exp is the exponent. To multiply the number by itself, the multiplication function was used, which implements the Karatsuba algorithm and is hence faster than the traditional brute force approach.