Bigint Calculator

A brief introduction to the implementation

Introduction

Bigint calculator calculate big numbers that you can't hold with built-in types in C++. Bigint keeps these large numbers as digits maxed out at 255 digits in an array data structure.

Bigint Class

Operators as public interfaces;

- Operator + for additions of bigint numbers together,
- Operator - for subtractions of bigint numbers,
- Operator / for dividends of bigint numbers and,
- Operator * for the multiplications between bigint numbers.

Public operators for inputting and outputting;

- Operator << for outputting a Bigint number on the console,
- Operator >> for inputting a Bigint number from a console user input.

Private fields and functions;

- *digits* to store the value of Bigint,
- *size* to store the size of Bigint,
- *setBigint()* takes an char array as a parameter and sets into the Bigint digits array,
- *freeMemory()* frees up the Bigint memory.

Operator +

The operator + takes left and the right operands and returns the sum of the two. Function removes any leading zeros to avoid the Bigint being unnecessary long.

Function iterates over the digits array of both left and right operands starting from the end of the array and adding up digit by digit and managing a carry amount to forward the extra 10th value to the next add. Time complexity is O(n).

Operator -

The operator - takes left and the right operands and returns the subtraction.

Function removes any leading zeros to avoid the Bigint being unnecessary long. Then, it decides what's the largest and smallest number and decides if the result is going to be a minus or a plus.

Function iterates over the digits array of both left and right operands starting from the end of the array and subtracting digit by digit and managing a carried amount to forward the 10th value to be subtracted from the upcoming left operand digit if any. Time complexity is O(n).

Operator *

The operator * takes its left and right operands as parameters. Multiplication is implemented by separating its functionality through out different functions for different jobs.

- *multiplyByDividendsOfTen()* multiplies the left operand Bigint from a digit from right operand.
- *addSameNumber()* recursively adds same number a number of times using the operator + and returns the final summation.

Operator * uses operator + for multiplying the two Bigint numbers.

Operator /

The operator / takes its left and right operands as parameters. Uses the operator - and operator + for the calculation.

The function decides what's the largest and smallest number and decides if the result is going to be zero or not. If the left operand is smaller than the right operand, function returns zero as the answer directly.

Function keeps subtracting the right operand from the left operand until it hits a minus number and then uses number of subtractions to calculate the dividend of two Bigint numbers.

Operator <<

The operator << takes the Bigint to be outputted as a parameter. Function removes any leading zeros from the Bigint and then outputs it digit by digit to avoid printing out any garbage value.

Operator >>

The operator >> takes the Bigint that should put the user input into as a parameter. Function takes an input from the user and set it to the Bigint using *setBigint()* function. Inputs are limited to be 255 chars which is the maximum size of Bigint.

Bigint Data Structure

Bigint uses a char typed array data structure named digits to hold its digit values as. Digits array has a maximum length of 255 chars.

First, the Bigint was implemented using a Stack data structure because the calculations included in Bigint are close to operations like push and pop and seemed always pop out the last inputted value. But the implementation got complex and in some places it needed to be able to see more than the last inputted value without popping out values. So, the implementation was changed to a more simple data structure Array from Stack.

Reasons that an array is used as the Bigint data structure are;

- It's the simplest data structure in use and easy to implement,
- Easy to manipulate,
- Reduces the risk of leaking memory since unlike using pointers with memory allocations, the memory management of array types are handled by the C++ itself,
- Input into an array is comparatively fast and takes only a time complexity of O(1),
- Retrieval is also fast and takes only a time complexity of O(1) if knows what index to retrieve and here in the Bigint calculator, it always knows what to retrieve from the digits array.

Bigint has carefully freed up all the memory allocations and used built-in arrays where possible so that it can reduces the risk of leaking memory.