Infinite Precision Arithmetic

Team 7
Nenavath kalyan nenavathkalyancs11039
Dheeraj Agarwal Dheerajagarwal1234

April 21, 2024

1 Introduction

The `Integer` class in C++ is used to work with large integers and floating numbers, which are numbers with many digits. Here's how it works:

2 Integer Class

2.1 Constructors and Destructor

The class has special functions called constructors and a destructor. Constructors are used to create new `Integer` objects, and the destructor cleans up memory when an object is destroyed.

3 Utility Functions

3.1 reverse

This function takes a string and flips it around.

3.2 padLeadingZeros

This function adds zeros to the beginning of a string to make it a certain length. For example, if you have the string "123" and you want it to be 5 characters long, it will become "00123".

3.3 compareStrings

This function compares two strings based on their lengths. It tells you which string is longer, or if they're the same length.

4 Arithmetic Operations

4.1 Addition (operator+)

This operation adds two `Integer` objects together. It can handle cases where both numbers are positive, both are negative, or one is positive and one is negative. If one number is negative, it converts the addition to subtraction.

4.2 Subtraction (operator-)

This operation subtracts one `Integer` from another. It handles cases where both numbers are positive, both are negative, or one is positive and one is negative. If one number is negative, it converts the subtraction to addition.

4.3 Multiplication (operator*)

This operation multiplies two `Integer` objects together. It handles cases where both numbers are positive, both are negative, or one is positive and one is negative. If one number is negative, it stores the result with a negative sign.

4.4 Division (operator/)

This operation divides one `Integer` by another. It handles division by zero and determines the sign of the result based on the signs of the dividend and divisor.

5 Design Section

5.1 Design Decisions

1. **Representation of Large Integers**:
   - Large integers are represented as strings rather than built-in integer types. This allows handling integers of arbitrary length without overflow issues.
   - The `Integer` class encapsulates this representation and provides arithmetic operations on large integers.
2. **Utility Functions**:
   - Utility functions such as `reverse`, `padLeadingZeros`, and `compareStrings` are provided to assist with arithmetic operations.
3. **Operator Overloading**:
   - Operators `+`, `-`, `*`, and `/` are overloaded to perform addition, subtraction, multiplication, and division operations on `Integer` objects.
   - The `operator<<` is overloaded to allow easy output of `Integer` objects.

4. **Arithmetic Operations**:

   - Addition and subtraction operations handle cases where both numbers are positive, both are negative, or one is positive and one is negative. If one number is negative, it converts the operation to addition.
   - Multiplication handles cases where both numbers are positive, both are negative, or one is positive and one is negative. It stores the result with the appropriate sign.
   - Division handles division by zero and determines the sign of the result based on the signs of the dividend and divisor.

6 Float Class

6.1 `removeLeadingZeros`

This function removes leading zeros from a string.
It iterates through the string and erases leading zeros until it encounters a non-zero digit.
If the string becomes empty after removing leading zeros, it sets it to "0".
Returns the modified string.

6.2 `removeTrailingZeros`

Similar to `removeLeadingZeros`, this function removes trailing zeros from a string.
It iterates through the string backward and erases trailing zeros until it encounters a non-zero digit.
If the string becomes empty after removing trailing zeros, it sets it to "0".
Returns the modified string.

6.3 `Float::parse`

This method is used to parse a string and create a Float object from it.
It takes a string as input and returns a Float object.
Internally, it constructs a Float object by splitting the input string into integer and fractional parts, then removing leading and trailing zeros from each part.

6.4 `Float::toString`

This method converts a Float object to its string representation. If the fractional part is "0", it returns only the integer part. Otherwise, it returns the concatenation of the integer part, a decimal point, and the fractional part.

6.5 padTrailingZeros

This function pads a string with zeros at the end to match a specified length. If the input string is shorter than the specified length, it appends the required number of zeros. Returns the padded string.

6.6 padLeadingZeros

Similar to `padTrailingZeros`, this function pads a string with zeros at the beginning to match a specified length. If the input string is shorter than the specified length, it prepends the required number of zeros. Returns the padded string.

6.7 compareStrings

This function compares two numbers represented as strings. It first compares the lengths of the strings and returns 1 if the first string is longer, -1 if the second string is longer, or 0 if they have the same length. If the lengths are equal, it compares the characters of the strings from left to right and returns 1, -1, or 0 based on the comparison result.

7 Addition

7.1 operator+

In this first we check that the operands are negative or not.If any operand is negative then we make it a positive number. Based on the sign of the operands we decide to subtract or add their absolute values and their resultant sign.then we call the function `addStrings` which takes four arguments.

7.2 addStrings function

In this we add separately add the integer and fractional part of operands. To make the size of fractional part and integer part we add Zeros in front and backward respectively. If carryfromFractional is true then we add 1 in integer part of result and fractional part from 2 digit of fraction. we remove the leading and Trailing zeros from integer and fractional part.

7.3 addNumericStrings function

In this function we add two numbers with the process that we used manually then after for loop we add last carry in starting and updating the value of carryout that use for carryFronFractional. Finally we return the value of result.

Overall, we added the Fractional part and integer part of operands separately and by checking their signs and comparison them we decide their resultant sign.

8 Subtraction

Similarly as addition we subtract two numbers by using subtractStrings and subtractNumericStrings functions.

first we check their signs and numeric values and based on that we decide to add and subtract their numeric values.Then we add their integer and fractional part separately.

9 Multiplication

9.1 operator*

Firstly we check their sign. then we convert them in positive numbers and remove the decimal point and join the integer and fractional part.
we apply a condition that any number is zero then it return 0 directly.
further we calculate the total fractional length of result to decide the position of decimal point and sign of result.
Then we call the function multiplyStrings function.
After returning the result by this function we compare the size of result and total fractional length and based on that we decide the integer and fractional part of that result. And in last we remove unneccessary zeros and return the result.

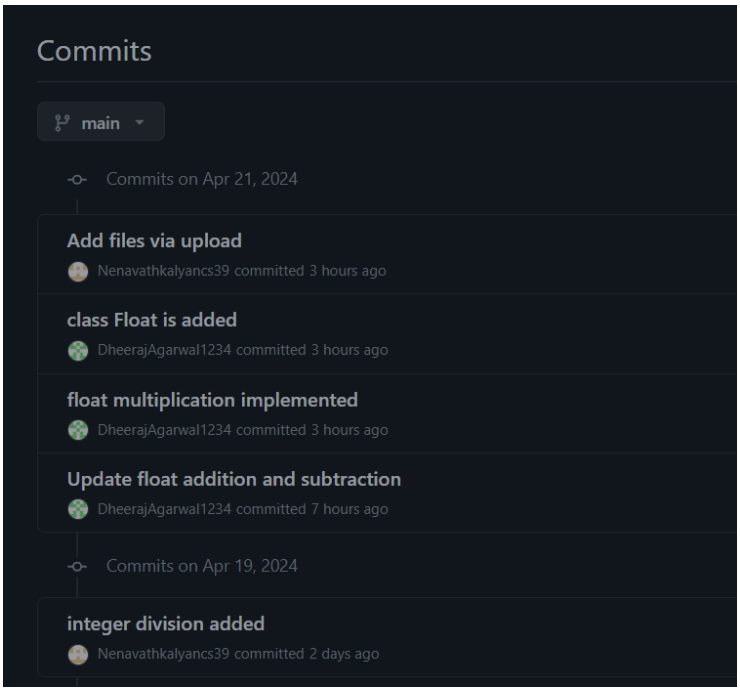
9.2 MultiplyStrings

Simply we calculate the product of two numbers manually and the resultant product to operator* function.

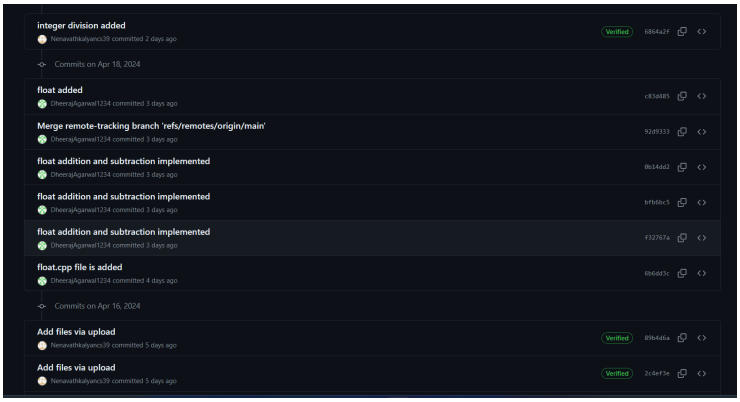
10 Verification approach

10.1 GDB

With the help of debugging we can verify our approach.



(a)



(b)

Figure 1: Git commits

Integer
operator+(const Float other) const
operator-(const Float other) const
operator*(const Float other) const
operator/(const Float other) const

Table 1: Integer Class

Float
operator+(const Float other) const
operator-(const Float other) const
operator*(const Float other) const
operator/(const Float other) const

Table 2: Float Class