Report.pdf

Member One
Tanishq Arvind Chirame
`Git ID:- ASST493`

Member Two
Bolla Lokesh Reddy
`Git ID:- Lokesh-Reddy011`

Date:-20 April 2024

1 Introduction

The Infinite Precision Arithmetic Library project aims to implement a versatile library for performing arithmetic operations on integers and floating-point numbers with arbitrary precision. The design involves organizing the functionality into well-structured classes and namespaces, facilitating modularity and extensibility.

2 Design Ideas

The project is structured around the following 3 key components:

2.1 .h file

Contains function prototypes and class declarations for integers (Integer) and floating-point numbers (Float). Ensures separation of interface from implementation.

2.2 Namespace

Inside the InfiniteArithmetic namespace, we have implemented the functions from the Integer and Float classes in IntegerIA.cpp and FloatIA.cpp respectively...

2.3 Key Features

2.3.1 Encapsulation

Class definitions achieve data hiding.
Access is controlled by using private and public access specifiers

2.3.2 Abstraction

Complex (or) Obscure implementations are hidden in a private section

2.3.3 Modularity

The specific class contains implementations specific to that class and hence supports modularity and helps in Improvisation (or) Updating aspects.

2.3.4 Proper Documentation

Proper comments are included for keeping track of the status of member variables etc.

3 UML Diagram

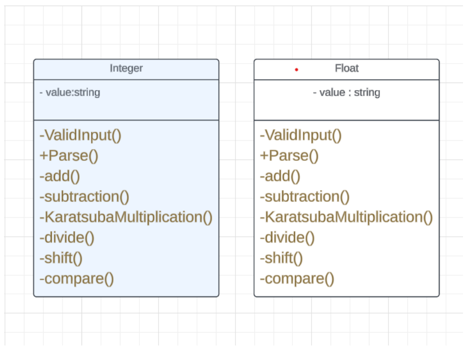
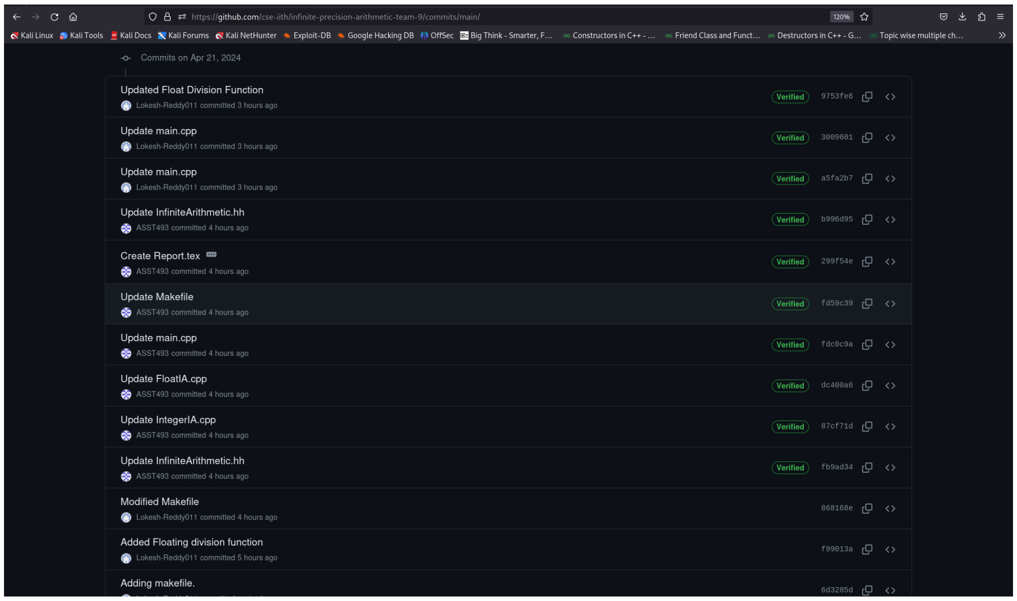


Figure 1: Enter Caption

4 Git Commits

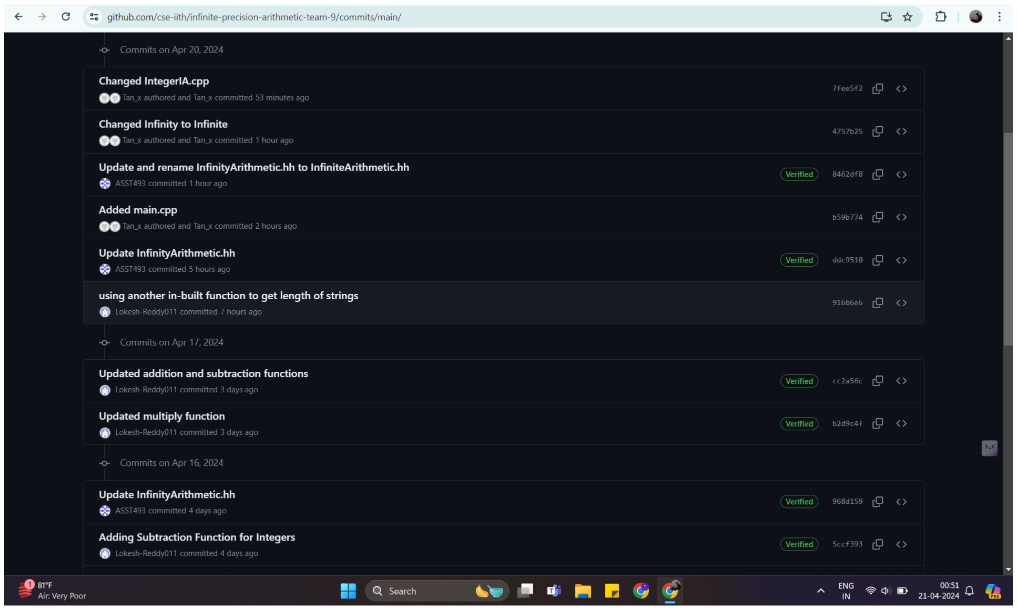


https://github.com/cse-iith/infinite-precision-arithmetic-team-9/commits/main/

Commits on Apr 21, 2024

```
Updated Float Division Function                       Verified  9753fe6
Lokesh-Reddy011 committed 3 hours ago
Update main.cpp                                       Verified  3009601
Lokesh-Reddy011 committed 3 hours ago
Update main.cpp                                       Verified  a5fa2b7
Lokesh-Reddy011 committed 3 hours ago
Update InfiniteArithmetic.hh                          Verified  b996d95
ASST493 committed 4 hours ago
Create Report.tex                                     Verified  299f54e
ASST493 committed 4 hours ago
Update Makefile                                       Verified  fd59c39
ASST493 committed 4 hours ago
Update main.cpp                                       Verified  fdc0c9a
ASST493 committed 4 hours ago
Update FloatIA.cpp                                    Verified  dc400a6
ASST493 committed 4 hours ago
Update IntegerIA.cpp                                  Verified  87cf71d
ASST493 committed 4 hours ago
Update InfiniteArithmetic.hh                          Verified  fb9ad34
ASST493 committed 4 hours ago
Modified Makefile                                               868168e
Lokesh-Reddy011 committed 4 hours ago
Added Floating division function                                f99013a
Lokesh-Reddy011 committed 5 hours ago
Adding makefile.                                                6d3285d
```

Figure 2: Git commits



github.com/cse-iith/infinite-precision-arithmetic-team-9/commits/main/

Commits on Apr 20, 2024

```
Changed IntegerIA.cpp                                           7fee5f2
Tan_x authored and Tan_x committed 53 minutes ago
Changed Infinity to Infinite                                    4757b25
Tan_x authored and Tan_x committed 1 hour ago
Update and rename InfinityArithmetic.hh to InfiniteArithmetic.hh   Verified  8462df8
ASST493 committed 1 hour ago
Added main.cpp                                                  b59b774
Tan_x authored and Tan_x committed 2 hours ago
Update InfinityArithmetic.hh                          Verified  ddc9510
ASST493 committed 5 hours ago
using another in-built function to get length of strings        916b6e6
Lokesh-Reddy011 committed 7 hours ago
```

Commits on Apr 17, 2024

```
Updated addition and subtraction functions            Verified  cc2a56c
Lokesh-Reddy011 committed 3 days ago
Updated multiply function                             Verified  b2d9c4f
Lokesh-Reddy011 committed 3 days ago
```

Commits on Apr 16, 2024

```
Update InfinityArithmetic.hh                          Verified  968d159
ASST493 committed 4 days ago
Adding Subtraction Function for Integers              Verified  5ccf393
Lokesh-Reddy011 committed 4 days ago
```

Figure 3: Snapshot of Commits

An appropriate (or) necessary number of commits were made by both the team members, both by directly uploading on the GitHub website and also using git commands.

5 Limitations of our library

Our library comes with the great functionality but with some limits to them... Let us look at those one by one

5.1 Exponentiation

Our library lacks Exponentiating functions like calculating a^b

5.2 Modulo

Our library lacks Modulo operator which calculates the remainder based on % operations.

5.3 Logarithmic Operations

Our library cannot perform those types of operations as well.

5.4 Numbers starting with +

Deliberately we ask users to start positive numbers with their first significant digit and not with the + sign...(considering the ethics...)
As there is saying that
"Signs are not an exception to the rule." - Alex Montañez This quote is related to using a plus sign before positive numbers in the context of statistics and mathematics. It suggests that everything, including signs, follows certain rules and conventions, and should not be used arbitrarily or without consideration for their significance. Remembering this rule can help avoid errors and have more accurate calculations.

5.5 Terminating decimals

Terminating decimals after the thousand number of digits are rounded of to 1000 digits after the decimal according to the requirements of the software specifications. This reduces the precision but is important while considering the buffer overflow.

5.6 Irrational Numbers

This library also doesn't provide operations for the Irrational numbers.

6 Verification Approach

There were many errors of many types involved...Some are as follows

6.1 Include path

One of our team members who was using VS code as an editor was facing an error which was like
Unable to resolve configuration with compilerPath ”D:/Buffer/mingw64”. Using ”C:\MinGW\bin\gcc.exe” instead.
which was maybe caused by to wrong location of the MinGW setup in the machine... So we tested it on some other machine and it worked...

6.2 Using GDB

We took manual edge test cases and then checked them manually line by line using GDB –¿ using watchpoints, breakpoints etc.

6.3 Using cassert header

We used the assert macro from the ”cassert” header to check some conditions and then break the code, but the same functionality was then achieved by using some if-else and break statements...

6.4 Code for checking number of digits

We wrote a cpp code for counting the digits after the decimal which should be 1000 in number as specified in the objective of the project.

6.5 Tried using GMP library

We downloaded the .tar.lz extension file which indicates a compressed archive file that has been created using two different compression techniques: tar (for archiving multiple files into a single file) and lz (for compressing the archived file using the LZ compression algorithm). but we were not able to use it, it was a binary file ...
So we just checked the results for our manual test cases by using an online infinite precision calculator...

README:

Infinite Precision Arithmetic

Welcome to Infinite Precision Arithmetic Library! Here we provide you classes for implementing the arbitrary precision arithmetic for integers and floats...This library is assured to give you correct result upto 1000 digits after decimal point which is more than enough...

Acknowledgements

- GMP library
- Karatsuba Algorithm
- How to write good readme

API Reference

This library supports operations on both integers and floats:

1. Addition
2. Subtraction
3. Multiplication
4. Division Our library doesn't contain some advanced mathematical functions like modulo, eponentiation etc. Method used to structure the files:
   1) One header file which consists the prototypes of the function , defines the namespace called InfiniteArithmetic, and 2 classes called Integer and Float.
   2)Then there were 2 .cpp files which contained the implementations of the funtions which were prototyped inside the (.hh) file.
   3)Then there is a main file which manipulates the command line arguments to invoke the proper functions...

Deployment

To deploy this project, follow these steps:

```
git clone <repository_url>
cd InfinitePrecisionArithmetic
make
./executable (int/float) (add/sub/mul/div) number1 number2
```

Be careful while writing the command as command line arguments are manipulated to call (or invoke) the proper functions...

Documentation

Brief overview of implemented mathematical functions:

1. **Integer Addition:** Implemented using manual carry addition from right to left.

2. **Integer Subtraction:** Implemented using manual subtraction.

3. **Integer Multiplication (Karatsuba):** Utilizes the Karatsuba algorithm for efficient multiplication of large integers.

    - Karatsuba Algorithm:
        - Split the Numbers: Given x and y, split them into halves x_1, x_0 and y_1, y_0.
        - Let $x = x_1 \cdot 10^{n/2} + x_0$
        - Let $y = y_1 \cdot 10^{n/2} + y_0$
        -
        - $x \cdot y = (x_1 \cdot 10^{n/2} + x_0) \cdot (y_1 \cdot 10^{n/2} + y_0)$
        - $x \cdot y = x_1 \cdot y_1 \cdot 10^n + (x_1 \cdot y_0 + x_0 \cdot y_1) \cdot 10^{n/2} + x_0 \cdot y_0$
        - Recursive Multiplications:
            - Compute $z_2 = x_1 \times y_1$
            - Compute $z_0 = x_0 \times y_0$
            - Compute $z_1 = (x_1 + x_0) \times (y_1 + y_0)$ using recursive calls.
        - Combine Intermediate Results:
            - Calculate $x \times y = z_2 \times 10^n + (z_1 - z_2 - z_0) \times 10^{n/2} + z_0$

    We manipulate x1 and x0 such that they have same number of digits, same is the case with y0 and y1. Manipulation here means, we just multiply by 10 to get one more digit if the number of the digits is odd in 'x', and finally we divide answer by 10, same way we do it for 'y' also if it has odd number of digits...

4. **Integer Division:** Implemented similar to manual long division.

For implementing the Division, we can also look at Newton-Raphson method which is also a very interesting algorithm...

Authors

- ASST493
- Bolla Lokesh Reddy

7 Key Learnings from the overall project...

Many things were new on which we bumped upon...
some of those are as follows:

7.1 Mathematical Algorithms

7.1.1 Montgomery Multiplication:

A technique used for modular multiplication, particularly useful in cryptography and number theory applications.

7.1.2 Karatsuba Multiplication:

Calculates the product of 2 numbers in recursive way...

7.1.3 Newton-Raphson Division

Calculates the Division of 2 numbers in recursive way usin recurrence relation which is given by...
the recurrence relation for applying the Newton-Raphson method to find the reciprocal of a number y is:

$$x_{n+1} = 2x_n - yx_n^2$$

7.1.4 Newton-Raphson for Square-Root

Calculates the square root of the number in recursive way which is given by...

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right)$$

7.2 About GMP library

We came to know about GNU multiple precision library which was again doing same task as we were doing, It had some interesting algorithms to calculate billions of digits of π .
We came across the new extensions named .tar and.lz which used to compress files using new LZ algorithm which was based on space optimization and efficiency...

7.3 Output streams

7.3.1 Buffered ones

"cout" is buffered output stream...
A buffered output stream writes data to an intermediate buffer in memory before flushing it to the target destination.

7.3.2 Unbuffered ones

”cerr” and ”clog” are unbuffered output streams...
An unbuffered output stream writes data directly to the target destination as soon as the write() operation is called.

7.4 The ’all-of’ function

The all-of function is a useful algorithm provided by the C++ Standard Library as part of the ¡algorithm¿ header. It is used to determine if a specified condition holds true for all elements within a given range defined by iterators.

7.5 Using +’0’ and -’0’

We can use +’0’ to convert integer to character...
We can use -’0’ to convert character to integer...