Infinite Precision report

Ponguru Bhanu

April 2024

Infinite Precision report

1 Project done by

1.1 Team 4

Member Role number	Member name
CS23BTECH11046	Ponguru Bhanu
CS23BTECH11047	Ponnamanda Manoj Kumar

Table 1: Team members

2 Design

made a namespace named InfinitePrecision to store everything related to the library.

2.1 Integer

2.1.1 constructor and overall representation

Integer is represented using a vector of digits. we used signed character to store each digit. a boolien for representing the sign of the integer. false for + and true for minus. we named the digits vector "digits" and sign boolien "sign".

given an string, we converted the string to integer by looping through the string and appending the digit given in each character of the string to the digits vector. we also checked the first character if any sign is given. fi there is a minus sign, then we will take that and update the sign value to 1. we used 0/1 and true/false for sign boolien in suitable places.

the default constructor initializes the integer to 0. that is, appending a 0 to the digits and leaving the sign as 0, that is +.

we also made the constructor support int family datatypes. that is short,long etc by using long long datatype, as that is the largest int family datatype. we first created a duplicate variable to store the absolute value of the given integer.

we looped through and appended the mod10 (reminder when divided by 10) of the number to the digits vector, followed by dividing the number with 10. we finally reversed the digits vector since we appended the mod10s in reverse order. finally setted the sign boolien to given number ¡ 0, that is 1 if negative, 0 otherwise.

2.1.2 addition/subtraction

to make it easy for future use, we first implemented a += operator.

first, to avoid any axidental change to the current object, we duplicated the digit vectors of 2 operands. we reversed the 2 digits. now we delt with 2 cases. if signs are equal for both the numbers, we added by following way.

first, we looped through the digits. we added the i_{th} position digits and stored it. we took the mod 10 of the current value as the result of i_{th} position of the result, and we sent the value divided by 10 to the $i + 1_{th}$ position of the result as carry. we then summed the obtained i_{th} position result to the result, since it will include any carrys obtained from $i - 1_{th}$ positions. we finally will obtain a new digits vector, and we set the sign to the sign of 2 given integers.

in the other case, that is when signs are opposit, we took integer having greater absolute value and subtracted with smaller absolute value. then used the sign of number which is having greater absolute value. the subtraction is as follows.

we first retrieved the subtracted value of i_{th} positions and carry will be -1 if the result obtained is negative, 0 otherwise. it uses the borrowing concept from pen papermethod, but in different way. carry will be -1 since we borrow from previous digit. we also add +10 to the result if obtained result is negative. we construct a digits vector by following this method.

finally, we use the digits vector obtained and sign to construct/change the current object with new digits and sign.

we also implemented a $-n$ equivalent for the integer. we simple used the + operator for implementing subtraction by $-n$ equivalent operator. given an integer n, the operator changes the sign of the n and returns.

also for the case of different length digit inputs, we simply appended 0s to pad the vectors to equal lengths.

2.1.3 multiplication

we first created an integer result with 0 value. then we duplicated the digits, again to avoid axidental changes to the current object.

we first looped through all the digits of a chosen first digits vector d1, say the iteration counter is i. for each iteration, a new vector res is created, to store the multiplyed value between i_{th} digit of d1 and d2. next, a nested for loop looping through d2 is created. each digit in d2 is multiplyed with current digit of d1. after multiplying 2 digits, similor to addition, a carry and result is stored. carry is integerr divided by 10 and result is mod10 of the multiplication. similor to the addition process, obtained result digit is added to the current position, since it also includes carrys from previous positions. we constructed a new res vector, and add that to the result integer created earlyer. we iterate the process for all the digits in d1, we multiply the res by 10 i times, i is number of iterations passed. since for each iteration, we need to shift the res digits while finally adding to the result. we used xor operator to get the sign of the new integer after multiplication. since it gives 1 for different booliens and 0 for same booliens. we finally construct/change the current object with obtained result.

2.1.4 division

for division, we took each digit in the numerator, looped through the all posible quotients, that is $0, 1, 2, ...$, untill we find a greatest quotient which makes denominator times itself less than or equal to the numerator. we find the reminder by simple subtraction, multiply it by 10, adding it to next digit and taking the resulting one to the next iteration. the iteration continues untill we run out of digits in numerator.

we also truncated extra zeros in the beginning, which are posible when first digit is less than denominator, and posibly some of the digits.

2.1.5 extra implementations

we also implemented overloaders for all comparison operators, a function to compute absolute value. we only implemented these because they are required for some implementation of main required functions.

2.2 Float

float is implemented mostly using the functionality of Integer, with few more implementations.

2.2.1 constructor and overall representation

float is represented using 2 integers. digits, and e to represent the float using $digits * 10^e$. we decided to use the same representation.

given a string, first we count all the digits after a '.' and store it in e as negative of that count. and we store all the numerical digits in to digits integer.

now, we implemented a constructor for double. we counted all the decimul places by subtracting the truncated version of the given number and original number. it gives the exact number of digits after decimul point. we now used all the digits and constructed the float using the obtained results.

by default, e and digits are 0, so the default constructor makes it 0.

2.2.2 addition,subtraction, and multiplication

given 2 floats, in case of addition, we first make sure that both e values are same. if not, we will decrement e value and multiply digits integer by 10 in a loop. after 2 e values became same, we will add the digits integer, since operators are already implemented in integer. we obtain the results and e value, construct/change the current object to the obtained result.

for the multiplication, we simply multiplyed digit integers, and added the e values to get new digits and e integers.

2.2.3 division

for the division, we used the following process.

we seperetly take numerator and denominator. we divide using the already implemented integer division. we first obtain a result. we will store it. then, if there is any reminder, we will loop through the division process untill we get a 0 reminder or we run out of precise digits. we multiplly the reminder by 10, decrease the exponent e by 1, and divide again, add it to teh obtained result multiplyed by 10. we obtain a new result. then we will get a new reminder and the process iterates untill we obtain 0 reminder or number of precise digits reached.

2.2.4 Extra implementations

all the extra functions implemented in integer are implemented in float too, as they are used in some main implementations.

2.3 UML diagram of float and integer

3 readme

3.1 using the executable

3.1.1 building the executable from source

use make to build the executable my$_{i}nf_{a}rith$

```
make
```

it makes a build folder with output files, and an executable named my$_{i}nf_{a}rith$

3.1.2 usage

use the program by

```
./my_inf_arith <dtype> <add/sub/mul/div> operand1 operand2
```

here, datatype is float or int. operand1 and operand2 are 2 numbers that you want to add. you can also set number of precise digits while division in the my$_{i}nf_{a}rith.cppfilebyset_{p}recisionfunction.pleasenotethatlargernumberofprecisedigitswilltakelotsoftimefo$

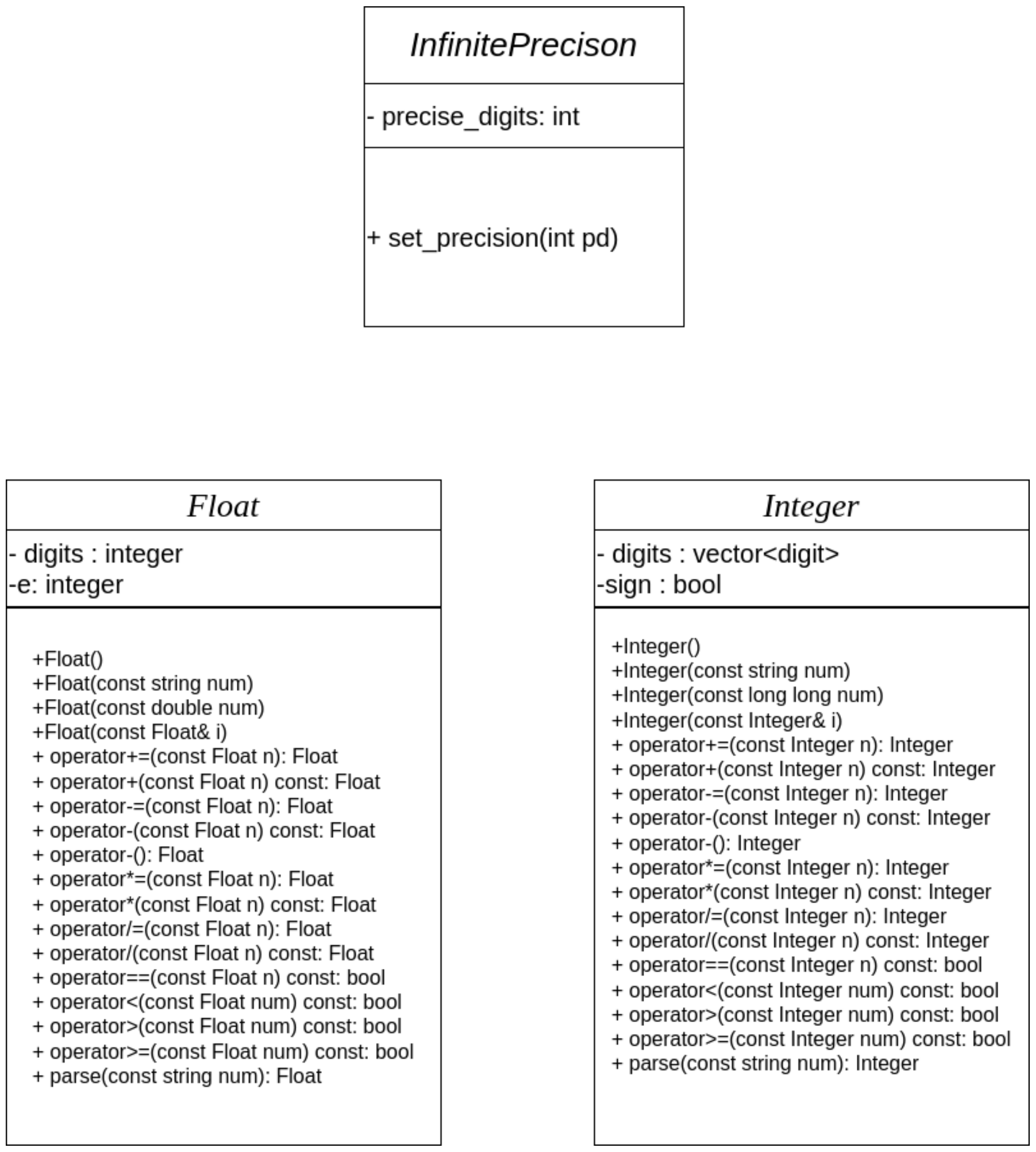

InfinitePrecison
- precise_digits: int
+ set_precision(int pd)
Float
- digits : integer
-e: integer
+Float()
+Float(const string num)
+Float(const double num)
+Float(const Float& i)
+ operator+=(const Float n): Float
+ operator+(const Float n) const: Float
+ operator-=(const Float n): Float
+ operator-(const Float n) const: Float
+ operator-(): Float
+ operator*=(const Float n): Float
+ operator*(const Float n) const: Float
+ operator/=(const Float n): Float
+ operator/(const Float n) const: Float
+ operator==(const Float n) const: bool
+ operator<(const Float num) const: bool
+ operator>(const Float num) const: bool
+ operator>=(const Float num) const: bool
+ parse(const string num): Float
Integer
- digits : vector<digit>
-sign : bool
+Integer()
+Integer(const string num)
+Integer(const long long num)
+Integer(const Integer& i)
+ operator+=(const Integer n): Integer
+ operator+(const Integer n) const: Integer
+ operator-=(const Integer n): Integer
+ operator-(const Integer n) const: Integer
+ operator-(): Integer
+ operator*=(const Integer n): Integer
+ operator*(const Integer n) const: Integer
+ operator/=(const Integer n): Integer
+ operator/(const Integer n) const: Integer
+ operator==(const Integer n) const: bool
+ operator<(const Integer num) const: bool
+ operator>(const Integer num) const: bool
+ operator>=(const Integer num) const: bool
+ parse(const string num): Integer

Figure 1: UML diagram

3.2 library

3.2.1 building the library libmy$_inf_arith.afromsource$

we can build the library by using the makefile again.

```
make libmy_inf_arith
```

you will get a libmy$_inf_arith.afile.youcanusethegiveninterfacefilestoaccessthelibraryfunctions.youcanlinky$
$+codewiththegeneratedlibrary.$

the usage is very similor to that of standard c++ datatypes. one limitation is that direct operator for ¿¿ is not implemented. so you can capture the stream in to string and use that string to construct objects of integer or float. almost all arithmetic operators, comparison operators are overloaded, except few like pre increment/decrement, post increment/decrement operators.

4 limitations

- a major limitation that we noticed later is that the algorithm we used for float division. the algorithm was intuitive but the time complexity became very large that it became almost imposible to get 1000 precise digits. so we setted the precise digits to 100, although 200 digits works but taking more time.
- another limitation is the ¿¿ operator. we did not implement the operator to read the input stream.
- time: for very large numbers, the time taken for a computation is significantly large.

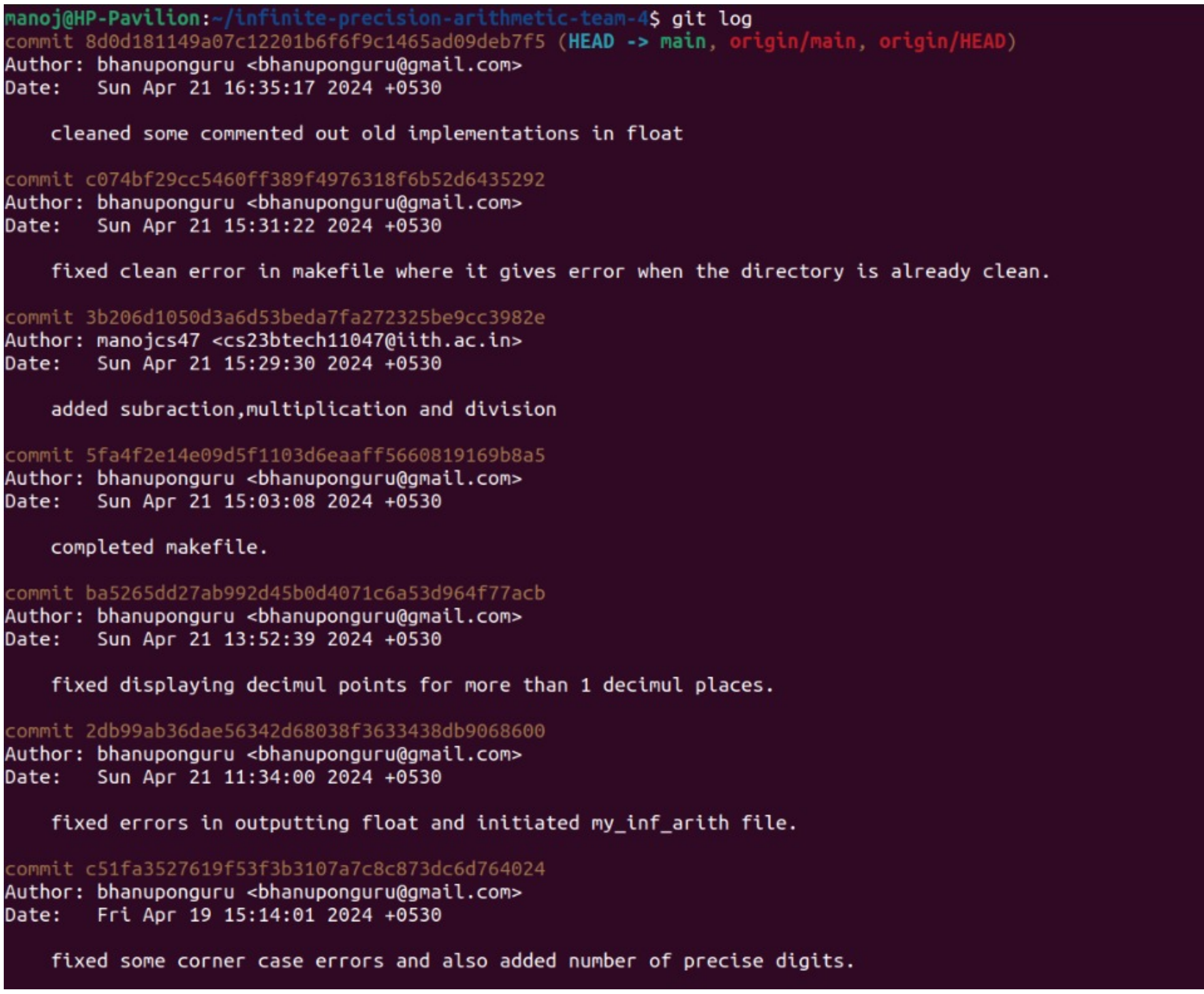
5 Verification approach

before we created code for my$_inf_arith,wefirstcreatedatest.cppandaddeditto.gitignore.wekeptmodifyingthatf$

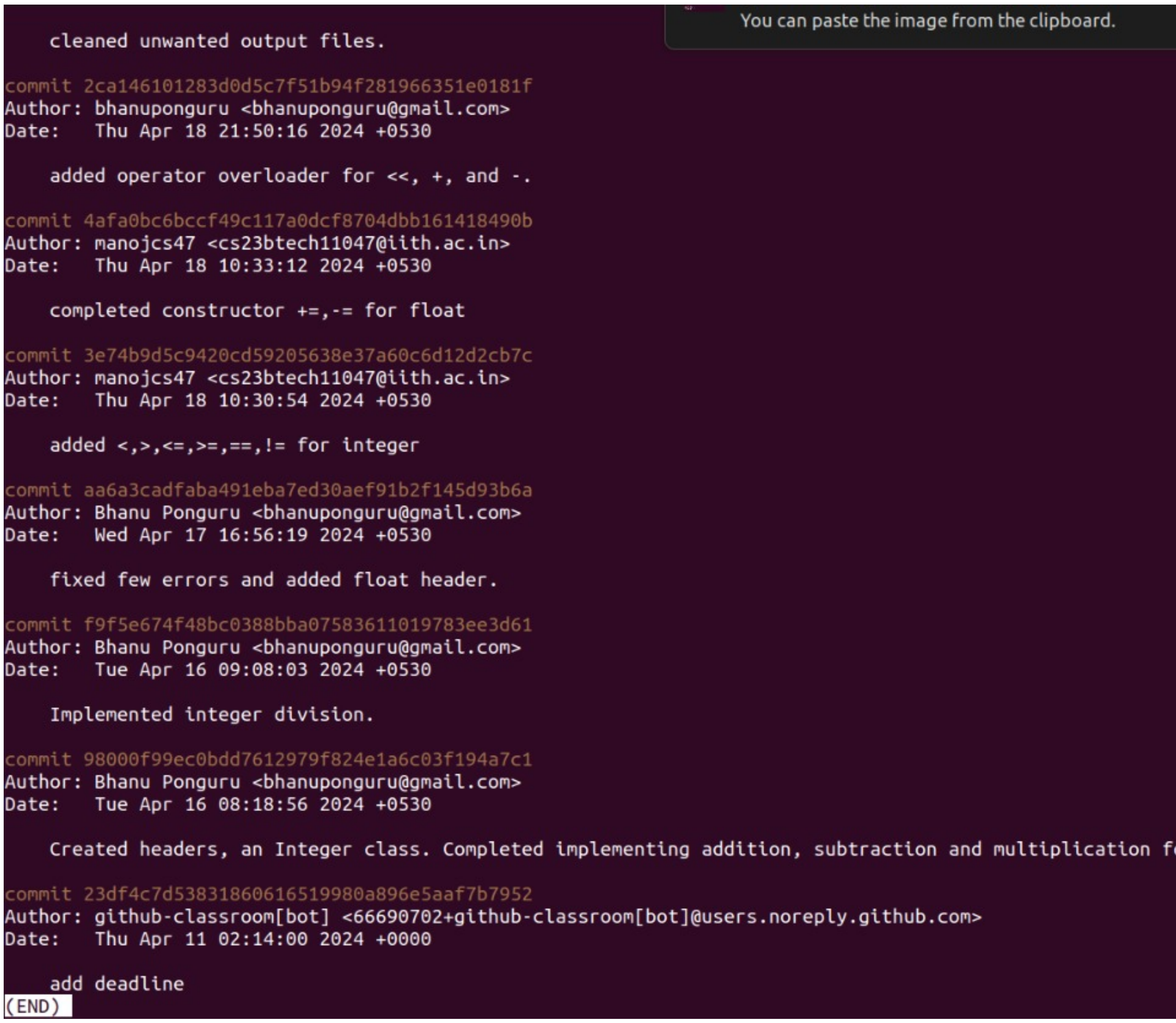
6 key learnings

one importent thing we learned is to test the time complexity of the algorithm while making the decision. we checked the intuitiveness of the float division algorithm, but we realised the significantly large time complexity later, after doing several test cases.

7 snapshots of git commits

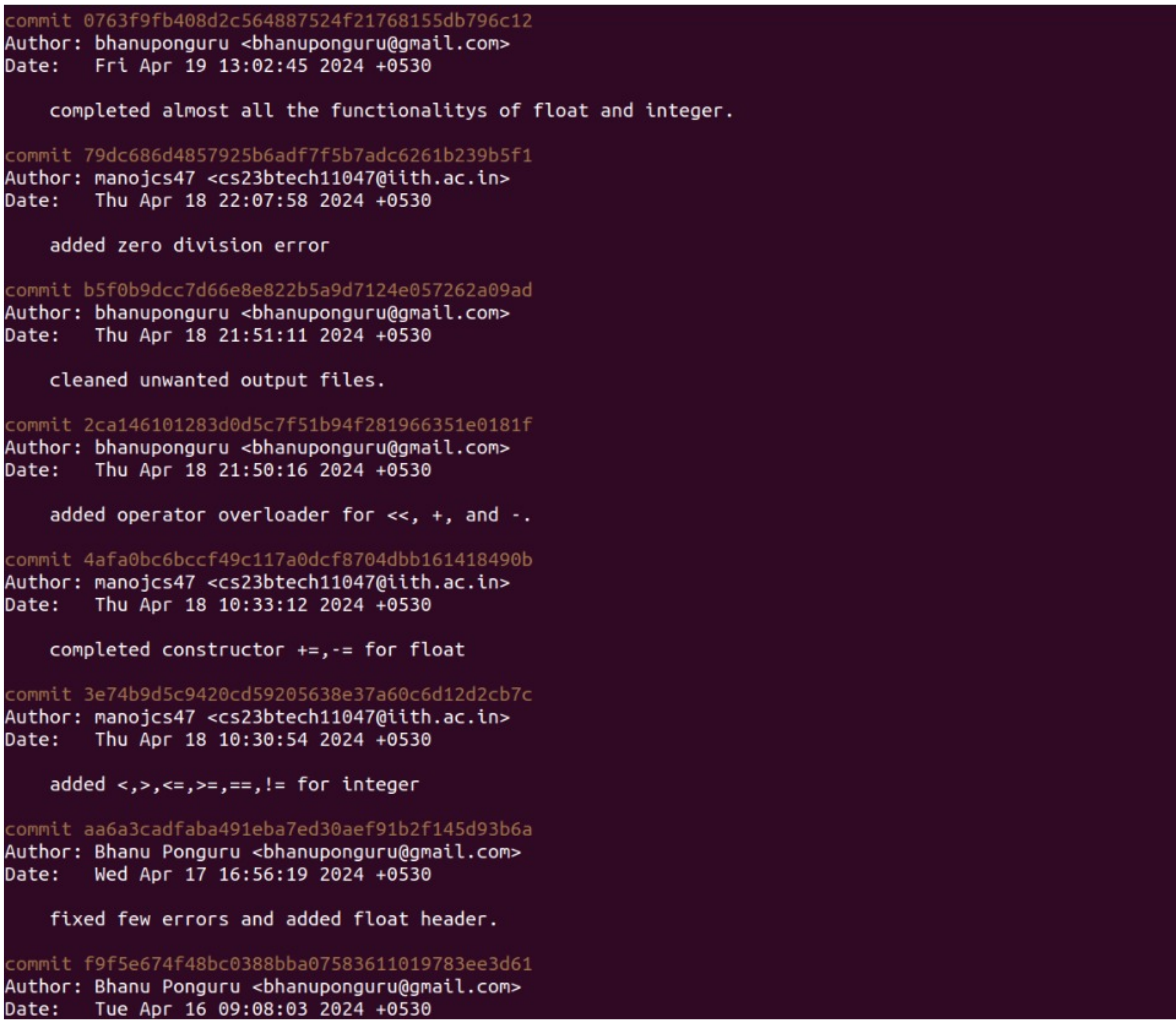


```
manoj@HP-Pavilion:~/infinite-precision-arithmetic-team-4$ git log
commit 8d0d181149a07c12201b6f6f9c1465ad09deb7f5 (HEAD -> main, origin/main, origin/HEAD)
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Sun Apr 21 16:35:17 2024 +0530

    cleaned some commented out old implementations in float

commit c074bf29cc5460ff389f4976318f6b52d6435292
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Sun Apr 21 15:31:22 2024 +0530

    fixed clean error in makefile where it gives error when the directory is already clean.

commit 3b206d1050d3a6d53beda7fa272325be9cc3982e
Author: manojcs47 <cs23btech11047@iith.ac.in>
Date:   Sun Apr 21 15:29:30 2024 +0530

    added subraction,multiplication and division

commit 5fa4f2e14e09d5f1103d6eaaff5660819169b8a5
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Sun Apr 21 15:03:08 2024 +0530

    completed makefile.

commit ba5265dd27ab992d45b0d4071c6a53d964f77acb
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Sun Apr 21 13:52:39 2024 +0530

    fixed displaying decimul points for more than 1 decimul places.

commit 2db99ab36dae56342d68038f3633438db9068600
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Sun Apr 21 11:34:00 2024 +0530

    fixed errors in outputting float and initiated my_inf_arith file.

commit c51fa3527619f53f3b3107a7c8c873dc6d764024
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Fri Apr 19 15:14:01 2024 +0530

    fixed some corner case errors and also added number of precise digits.
```



You can paste the image from the clipboard.

```
    cleaned unwanted output files.

commit 2ca146101283d0d5c7f51b94f281966351e0181f
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Thu Apr 18 21:50:16 2024 +0530

    added operator overloader for <<, +, and -.

commit 4afa0bc6bccf49c117a0dcf8704dbb161418490b
Author: manojcs47 <cs23btech11047@iith.ac.in>
Date:   Thu Apr 18 10:33:12 2024 +0530

    completed constructor +=,-= for float

commit 3e74b9d5c9420cd59205638e37a60c6d12d2cb7c
Author: manojcs47 <cs23btech11047@iith.ac.in>
Date:   Thu Apr 18 10:30:54 2024 +0530

    added <,>,<=,>=,==,!= for integer

commit aa6a3cadfaba491eba7ed30aef91b2f145d93b6a
Author: Bhanu Ponguru <bhanuponguru@gmail.com>
Date:   Wed Apr 17 16:56:19 2024 +0530

    fixed few errors and added float header.

commit f9f5e674f48bc0388bba07583611019783ee3d61
Author: Bhanu Ponguru <bhanuponguru@gmail.com>
Date:   Tue Apr 16 09:08:03 2024 +0530

    Implemented integer division.

commit 98000f99ec0bdd7612979f824e1a6c03f194a7c1
Author: Bhanu Ponguru <bhanuponguru@gmail.com>
Date:   Tue Apr 16 08:18:56 2024 +0530

    Created headers, an Integer class. Completed implementing addition, subtraction and multiplication f

commit 23df4c7d53831860616519980a896e5aaf7b7952
Author: github-classroom[bot] <66690702+github-classroom[bot]@users.noreply.github.com>
Date:   Thu Apr 11 02:14:00 2024 +0000

    add deadline
(END)
```


```
commit 0763f9fb408d2c564887524f21768155db796c12
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Fri Apr 19 13:02:45 2024 +0530

    completed almost all the functionalitys of float and integer.

commit 79dc686d4857925b6adf7f5b7adc6261b239b5f1
Author: manojcs47 <cs23btech11047@iith.ac.in>
Date:   Thu Apr 18 22:07:58 2024 +0530

    added zero division error

commit b5f0b9dcc7d66e8e822b5a9d7124e057262a09ad
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Thu Apr 18 21:51:11 2024 +0530

    cleaned unwanted output files.

commit 2ca146101283d0d5c7f51b94f281966351e0181f
Author: bhanuponguru <bhanuponguru@gmail.com>
Date:   Thu Apr 18 21:50:16 2024 +0530

    added operator overloader for <<, +, and -.

commit 4afa0bc6bccf49c117a0dcf8704dbb161418490b
Author: manojcs47 <cs23btech11047@iith.ac.in>
Date:   Thu Apr 18 10:33:12 2024 +0530

    completed constructor +=,-= for float

commit 3e74b9d5c9420cd59205638e37a60c6d12d2cb7c
Author: manojcs47 <cs23btech11047@iith.ac.in>
Date:   Thu Apr 18 10:30:54 2024 +0530

    added <,>,<=,>=,==,!= for integer

commit aa6a3cadfaba491eba7ed30aef91b2f145d93b6a
Author: Bhanu Ponguru <bhanuponguru@gmail.com>
Date:   Wed Apr 17 16:56:19 2024 +0530

    fixed few errors and added float header.

commit f9f5e674f48bc0388bba07583611019783ee3d61
Author: Bhanu Ponguru <bhanuponguru@gmail.com>
Date:   Tue Apr 16 09:08:03 2024 +0530
```