Report

Sai Rishi Kasturi cs23btech11024
Muvvala Sairam Suhas cs23bbtech11038
Team 20

April 21, 2024

Contents

1 Design

We have written the code such that it is nicely encapsulated by using the private members in the class and have written it in an object oriented way as you can see after reading this.

- we used #pragma once in the beginning so that when we are using this header file again and again we wont have the problem of repeating the whole code again and again and get unexpected errors while doing so unintentionally.
- we have used 2 headers in order to use the built in functions present in them which are

```
1    #include <algorithm>
2    #include <iostream>
3
```

- we have written the whole code in the namespace Infinite_Arithmetic so that when we are using the objects of that class we wont be having trouble with objects of classes having same names while using them.
- we have written two classes which are having the names Integers and Float.

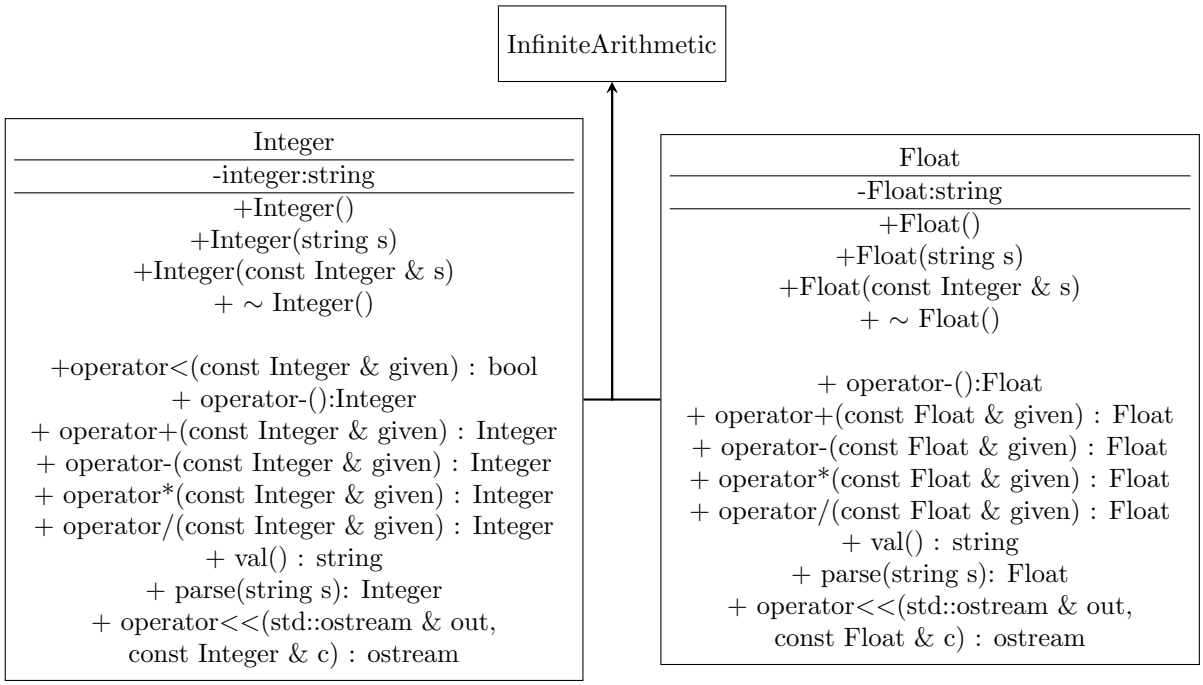
1.1 Integer

- we have used a private variable having the type string so that we can take inputs of very long values which cant be handles with our normal int datatypes.
- we have written a default Constructor a Constructor and a copy Constructor.we have also written a destructor.
- we have written Functions which overload various operators such as < - + * / -(2nd time).we have written a function which would make private variable int0 accessible to non class functions.we have also written friend function by overloading the cout operator in order to directly print the Integer object int0.we have used static function parse too which could create an Integer object from string without using the instance of a class

1.2 Float

- we have used the Constructor similar to that of Integer class just that we have also included a private integer called place whose value is the index of the decimal point in a float variable.
- The functions and operator overloading that we have done is similar to that of the Integer class

1.3 UML diagram



2 Readme

Infinite Arithmetic Library This library provides two classes which can be used to create large instances of Integers which cant be used using the basic datatypes and hence performing arbitary precision arithmetic operations on integers and floating point numbers created by the class.

2.1 Arbitary size

The integer class as well as the float class represents integers of arbitrary size without having an overflow or underflow.

2.2 Private variables

If we want to use the integer or the float value outside that of a class function we can do it by using a val function that is described in the library.

2.3 Outstream

If we want to print an Integer which is a private variable it is ok to directly use a cout because we have overloaded the << operator making it easy for us to print the outcome after using the overloaded functions.

2.4 Operator overloading

We can do addition subraction multiplication and division for various objects as we are returning the same object again after performing the operation and hence multiple operations between Integers or between Floats is possible.

2.5 Constructor

The Constructor has been made such that when not given any value the defaut Constructor is called and the values are initialized.

2.5.1 Integers

The Constructor is such that the leading zeroes are removed by itself when called using that Constructor.

```
Integer a("1234567");
Integer b = Integer::parse("-009876321");
\\leading 0s in b are removed
Integer c = a + b;
Integer d = a - b;
Integer e = a * b;
Integer f = a / b;
std::cout << "a = " << a << std::endl;
\\can directly print it
```

2.5.2 Floats

The Constructor when given a string only can itself remove all the trailing and leading zeroes and also find the place value of the decimal and initiates it. f we have been given a Float without a decimal point then it automatically adds a decimal point and a 0 after that so we dont have to worry about it being given an integer value

```
Float a("123.4567");
Float b = Integer::parse("-009876.3210");
\\leading and trailing 0s in b are removed
Float c = a + b;
Float d = a - b;
Float e = a * b;
Float f = a / b;
std::cout << "a = " << a << std::endl;
\\can directly print it
```

3 Limitations

3.1 Performance

the performance for the code of using the Floats and Integer classes are strings and using them take up a lot of memory in order to do the operations and hence the manipulations and memory allocation could slow the code down.

3.2 Exception handling

The integer class cant really handle the exception of being given the decimal point numbers if given they would treat the decimal point as a character or a digit by taking in its ASCII code.

3.3 Overloaded operators

There are only a few operators that have been overloaded and need more operators such as power == != > etc

3.4 Type conversions

We haven't been able to give explicit or implicit datatype conversions for the following classes.

4 Verification approach

- We have tested as we wrote the .h file while we wrote every function while we overloaded every operator even when we were writing the Constructor we checked weather the code was working properly as it is supposed to like when we were going through :-

4.1 Constructors

we checked weather the trailing 0s were being removed or not weather the leading 0s were weather the place value we were getting through the Constructor was correct or not weather the place value of the decimal was being given correct index or not.

4.2 Operators

with every case we checked we stumbled upon a part which we hadn't done in our algorithm which could make it better when faced with ambiguous values we would debug the code at every point using GDB and find out where the problem is and edit our code as such.

4.3 Short circuiting

After writing the operators algorithm we always checked weather we were being able to short circuit the code when we can use 0s anywhere possible.

5 Keylearnings

- we got to know how to keep testing and verifying the results by using GDB
- we came to know about various ways of how to write code,do proper indentation and also on how to comment properly
- we got to know how to write a parse function and a value function
- we learnt how difficult it is to find a algorithm no matter how small it is and came to understand the joy of finding one after thorough search for it

6 Git Snapshots

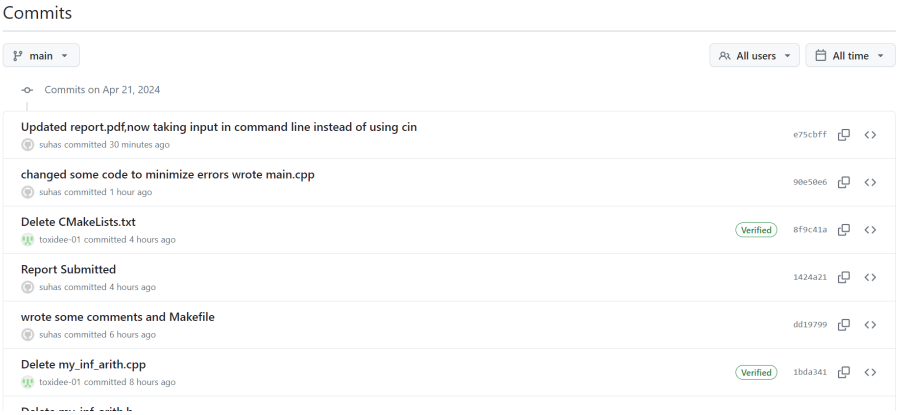


Figure 1: GIT COMMIT SCREENSHOTS

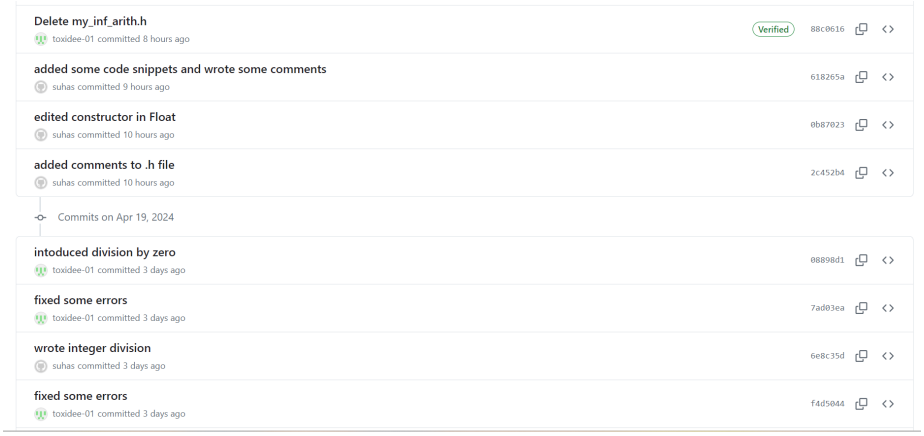


Figure 2: GIT COMMIT SCREENSHOTS

Figure 3: GIT COMMIT SCREENSHOTS

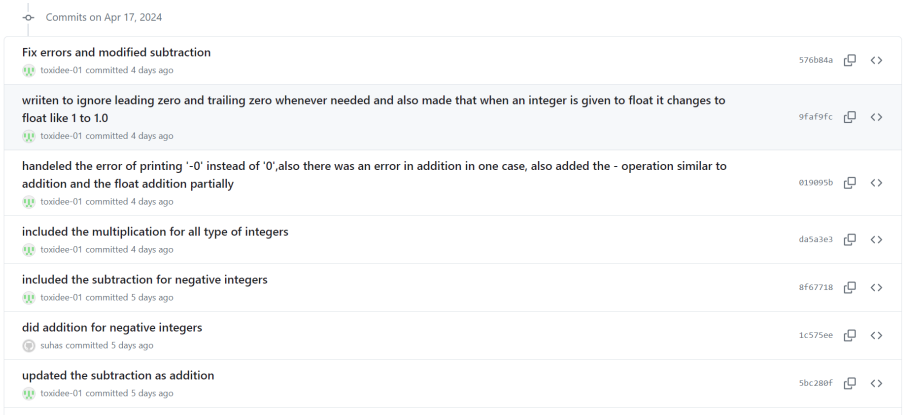


Figure 4: GIT COMMIT SCREENSHOTS

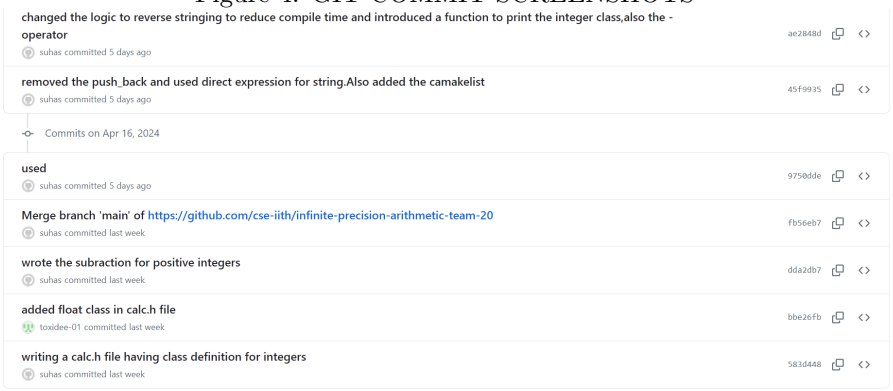


Figure 5: GIT COMMIT SCREENSHOTS

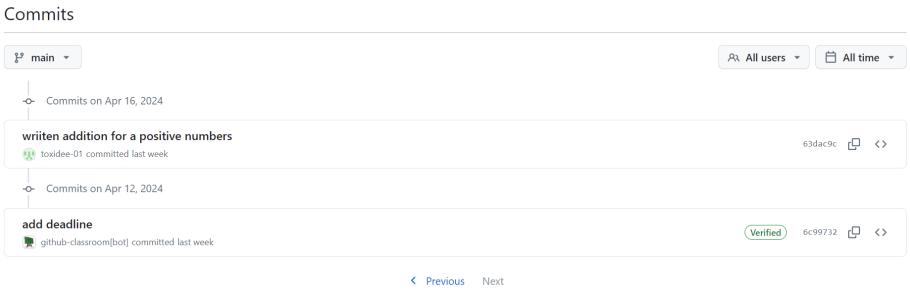


Figure 6: GIT COMMIT SCREENSHOTS