Infinite Precision Library Documentation

Sathvik Reddy Bhavanam

CS23BTECH11056

April 21, 2024

Contents

1 Installation

This section consisits of details regarding installation of the library.

1.1 Cloning The Repository

To clone the repository, use the following command:

```
git clone https://github.com/cse-iith/infinite-precision-arithmetic-team-12.git
```

Cloning the repository creates a local copy of all the files on your computer. You can then use this library to create your own code.

1.2 Building The Executable

```
make all
```

This creates a an executable that can run as your daily calculator.

1.3 Using The Executable

```
./my_inf_arith <type> <operation> <number1> <number2>
```

1.4 Building The Library

To create a library to link with your program run the following command:

```
make libmy_inf_arith
```

1.5 Linking The Library

Run the following command to link the library with file:

```
g++ <your-file-here>.cpp -o <executable> -std=c++17 -I include/ -L. -lmy_inf_arith
```

2 Design

2.1 Underlying Implementation

The data structure used for storing the numbers is a vector of 16 bit unsigned integers. Vector is probably the best data structure for doing so. This is because lists can be two slow. Strings were another option. I chose skip it because it is inconvient to use as I would have to subtract '0' everytime.

2.2 Limitations Of Vector

There aren't really many limitations of using vector. The only thing is when you pass a vector to a function. It takes more time to copy/assign.

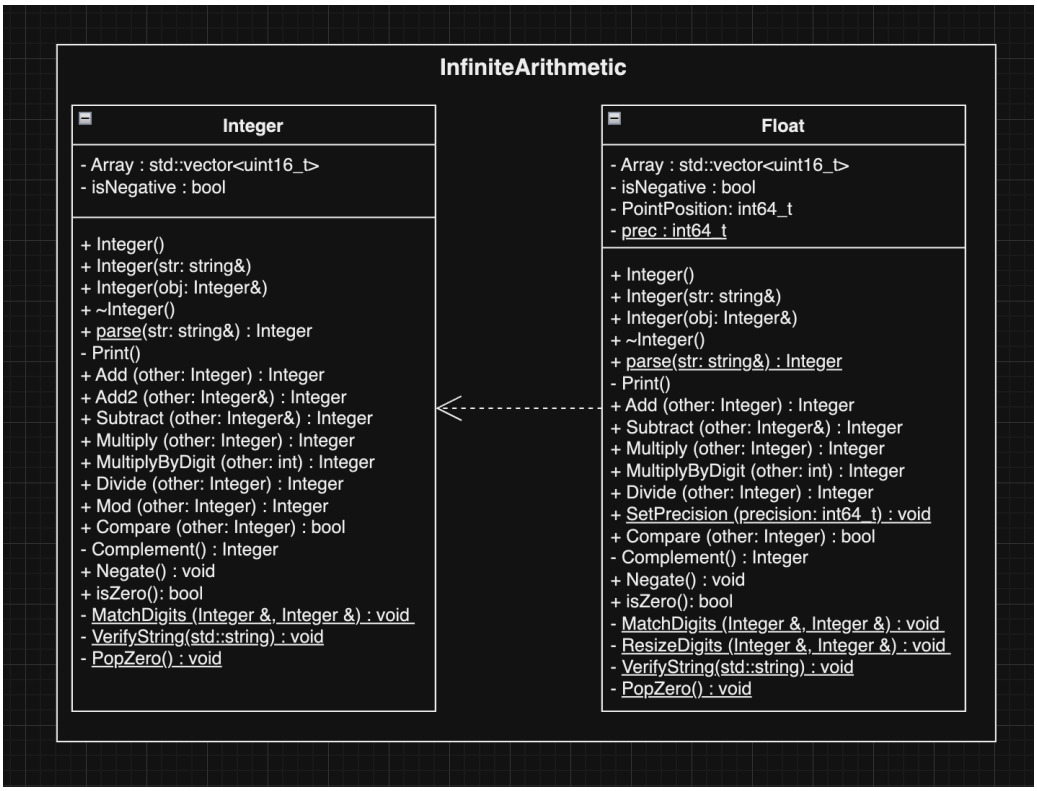
2.3 Error Handling

Input strings are parsed to check if they incorrect characters. Division by Zero is also handled appropriately.

2.4 Memory Management

Memory is completely managed by vector.

2.5 UML Diagram



3 API Tables

Note: the ones prefixed with @ are private methods.

3.1 Integer Table

Functions	Purpose
Assign	Used to assign the object to another value
Add	Adds two numbers together
Subtract	Subtracts one number from the other
Multiply	Multiplies two numbers
Divide	Divides two numbers
Negate	Negates the number
isZero	Tells if the number is zero or not
parse	Returns an instance of the Integer class
Compare	Compares the object to another and returns a value among `{0, 1, 2}`
Negate	Negates the object
@Complement	Takes the complement of given number
@MatchDigits	Matches the digits of two numbers such that they are equal
@PopZero	Removes redundant zeroes from the number
@Print	Displays the number

Operators	Purpose
operator<<	Used to assign the object to another value
operator>>	Used to take the number from input stream
operator=	Used to assign the object to another value
operator+	Adds two numbers
operator-	Subtracts one number from the other
operator*	Multiplies two numbers
operator/	Divides one number by the other
operator%	Takes the mod of one number w.r.t the other
operator+	Returns the same number
operator-	Negates the number
operator~	Takes the complement of the number

3.2 Float Table

Functions	Purpose
Assign	Used to assign the object to another value
Add	Adds two numbers together
Subtract	Subtracts one number from the other
Multiply	Multiplies two numbers
Divide	Divides two numbers
Negate	Negates the number
isZero	Tells if the number is zero or not
parse	Returns an instance of the Float class
Compare	Compares the object to another and returns a value among `{0, 1, 2}`
Negate	Negates the object
@Complement	Takes the complement of given number
@MatchDigits	Matches the digits of two numbers such that they are equal
@PopZero	Removes redundant zeroes from the number
@Print	Displays the number

Operators	Purpose
operator<<	Used to assign the object to another value
operator>>	Used to take the number from input stream
operator=	Used to assign the object to another value
operator+	Adds two numbers
operator-	Subtracts one number from the other
operator*	Multiplies two numbers
operator/	Divides one number by the other
operator+	Returns the same number
operator-	Negates the number
operator~	Takes the complement of the number

4 Integer Library

4.1 Introduction

Given below is an API reference for Integer:

4.2 API Reference

Here is a list of a functions that are a part of the `Integer` class.

4.2.1 Constructors

`Integer()`

This constructor creates a vector initializes it to `{0}` and initializes `isNegative` to `false`.

```
#include "Integer.h"
#include "Float.h"

int main()
{
  LOG(InfiniteArithmetic::Integer());
  // Output = 0
}
```

`Integer(std::string num)`

This constructor sets the member variables to appropriate values based on the string. It calls the `VerifyString` function internally to check if the string provided is valid or not.

```

  LOG(InfiniteArithmetic::Integer("212"));
  // Output = 212
.
```

`Integer(const Integer &obj)`

It is a copy constructor that replicates the values of the object given.

```

  namespace InfiniteArithmetic = IA;
  LOG(IA::Integer(IA::Integer("110")));
  // Output = 110
.
```

4.2.2 Destructor

`~Integer()`

The destructor deletes the vector and the boolean variable `isNegative` explicitly.

4.2.3 parse

`Integer parse(const std::string &)`

The parse function returns an instance of the `Integer` class.

4.2.4 Assign

`void Assign(Integer)`

The `Assign` function assigns the value of one integer to the other.

```

  IA::Integer num1 ("102");
  IA::Integer num2;
  num2.Assign(num1);
  LOG(num2);
  // Output = 102
.
```

The `= operator` is an other function that has been overloaded to assign one variable to another.

4.2.5 Print

The `Print` is a private function that is used to display the contents of the vector.
However, the `<<` operator (insertion operator) has been overloaded.

```

  using namespace InfiniteArithmetic;

  Integer num1("322");

  std::cout << num1  << std::endl;
  // Output = 322
.
```

4.2.6 Input

Input can be taken through `>>` operator (extraction operator).

```

  using namespace InfiniteArithmetic;

  Integer num1;
  Integer num2;

  std::cin >> num1 >> num2;
.
```

4.2.7 Add

`Integer Add(Integer)`

The `Add` function takes in two copies of `Integer` as it modifies them internally. It call the `MatchDigits` function to adjust the size of the inputs (for proper addition).
The `+` operator has been overloaded. It internally calls the `Add` function.

```

  using namespace InfiniteArithmetic;

  Integer num1 ("1023");
  Integer num2 ("3213");

  LOG(num1.Add(num2));
  // Output = 4236
.
```

4.2.8 Add2

`Integer Add2(Integer &)`

The `Add2` function has a small modification in that it is a little more effiecient(does 3 seconds better over 1000000 testcases). It takes in the variables as references and does not call the `MatchDigits` function.

```

  using namespace InfiniteArithmetic;

  Integer num1 ("1023");
  Integer num2 ("3213");

  LOG(num1.Add2(num2));
  // Output = 4236
.
```

4.2.9 Subtract

`Integer Subtract(Integer)`

The `Subtract` function negates the second number and calls the `Add` function.
The `-` operator has been overloaded. It internally calls the `Subtract` function.

```
  using namespace InfiniteArithmetic;

  Integer num1 ("1023");
  Integer num2 ("3213");

  LOG(num1.Subtract(num2));
  // Output = -2190
.
```

4.2.10 Multiply

`Integer Multiply(Integer)`

The `*` operator has been overloaded for multiplication.

```
  using namespace InfiniteArithmetic;

  Integer num1 ("322");
  Integer num2 ("4221");

  LOG(num1.Multiply(num2));
  // Output = 1359162
.
```

4.2.11 MultiplyByDigit

`Integer MultiplyByDigit(int)`

This function is called internally by `Divide`. It multiplies an `Integer` by a single digit`(0-9)`.

```

  using namespace InfiniteArithmetic;

  Integer num1 ("322");
  LOG(num1.MultiplyByDigit(2));
  // Output = 644
.
```

4.2.12 Divide

`Integer Divide(Integer)`

The function `Divide` internally does human long division. The `/` operator has been overloaded for division. Here's an example:

```
1
2   using namespace InfiniteArithmetic;
3
4   Integer num1 ("322");
5   Integer num2 ("22");
6
7   LOG(num1.Divide(num2));
8   // Output = 14
9 .
```

4.2.13 Mod

`Integer Mod(Integer)`

The `Mod` function returns the remainder after a division. It is calculated using the formula $p - (p/q) * q$ where p and q represent `Integer` objects.

```
1
2   using namespace InfiniteArithmetic;
3
4   Integer num1 ("322");
5   Integer num2 ("23");
6
7   LOG(num1.Mod(num2));
8   // Output = 14
9 .
```

4.2.14 Compare

`int16_t Compare(Integer)`

The `Compare` function returns a value in `{-1, 0, 1}` depending on which number is greater or lesser. All the comparison operators have been overloaded to do the same.

```
1
2   using namespace InfiniteArithmetic;
3
4   Integer num1 ("2");
5   Integer num2 ("5");
6
7   LOG(num1.Compare(num2));
8   // Output = -1
9 .
```

4.2.15 Complement

`Integer Complement() const`

The `Complement` function takes the complement of a number with respect to 9. For example, 102's complement is 898.
It is a private member function.

4.2.16 Negate

`Integer Negate()`

The `Negate` function changes the sign of the number. The unary `-` operator also does the same.

```
1
2   using namespace InfiniteArithmetic;
3
4   Integer num1 ("5");
5   LOG(num1.Negate());
6   // Output = -5
7 .
```

4.2.17 isZero

`bool isZero()`

The `isZero` function checks if the number is 0.

```
1
2   using namespace InfiniteArithmetic;
3
4   Integer num1 ("0");
5   LOG(num1.isZero());
6   // Output = 1
7
8   Integer num2 ("2");
9   LOG(num2.isZero());
10  // Output = 0
11 .
```

4.2.18 MatchDigits

`void MatchDigits(Integer &, Integer &)`

The `MatchDigits` is a static function that makes the vectors of the two numbers to equal length. This is a private function.

4.2.19 VerifyString

`void VerifyString(std::string)`

The `VerifyString` checks for any non-digit characters and raises an error if found. This is also a private function.

4.2.20 PopZero

`void PopZero()`

This function removes any redundant zeroes present in the number.

5 Float Library

5.1 Introduction

Given below is an API reference for Float:

5.2 API Reference

Here is a list of a functions that are a part of the `Float` class.

5.2.1 Constructors

`Float()`

This constructor creates a vector initializes it to `{0}` and initializes `isNegative` to `false`.

```
#include "Float.h"
#include "Float.h"

int main()
{
  LOG(InfiniteArithmetic::Float());
  // Output = 0
}
```

`Float(std::string num)`

This constructor sets the member variables to appropriate values based on the string. It calls the `VerifyString` function internally to check if the string provided is valid or not.

```

  LOG(InfiniteArithmetic::Float("212"));
  // Output = 212
.
```

`Float(const Float &obj)`

It is a copy constructor that replicates the values of the object given.

```

  namespace InfiniteArithmetic = IA;
  LOG(IA::Float(IA::Float("110")));
  // Output = 110
.
```

5.2.2 Destructor

`~Float()`

The destructor deletes the vector and the boolean variable `isNegative` explicitly.

5.2.3 parse

`Float parse(const std::string &)`

The parse function returns an instance of the `Float` class.

5.2.4 Assign

`void Assign(Float)`

The `Assign` function assigns the value of one Float to the other.

```

  IA::Float num1 ("102");
  IA::Float num2;
  num2.Assign(num1);
  LOG(num2);
  // Output = 102
.
```

The `= operator` is an other function that has been overloaded to assign one variable to another.

5.2.5 Print

The `Print` is a private function that is used to display the contents of the vector.
However, the `<<` operator (insertion operator) has been overloaded.

```

  using namespace InfiniteArithmetic;

  Float num1("322");

  std::cout << num1  << std::endl;
  // Output = 322
.
```

5.2.6 Input

Input can be taken through `>>` operator (extraction operator).

```

  using namespace InfiniteArithmetic;

  Float num1;
  Float num2;

  std::cin >> num1 >> num2;
.
```

5.2.7 Add

`Float Add(Float)`

The `Add` function takes in two copies of `Float` as it modifies them internally. It call the `MatchDigits` function to adjust the size of the inputs (for proper addition).
The `+` operator has been overloaded. It internally calls the `Add` function.

```

  using namespace InfiniteArithmetic;

  Float num1 ("203212.231");
  Float num2 ("42132.0322");

  LOG(num1.Add(num2));
  // Output = 245344.2632
.
```

5.2.8 Subtract

`Float Subtract(Float)`

The `Subtract` function negates the second number and calls the `Add` function.
The `-` operator has been overloaded. It internally calls the `Subtract` function.

```
  using namespace InfiniteArithmetic;

  Float num1 ("203212.231");
  Float num2 ("42132.0322");

  LOG(num1.Subtract(num2));
  // Output = 161080.1988
.
```

5.2.9 Multiply

`Float Multiply(Float)`

The `*` operator has been overloaded for multiplication.

```
  using namespace InfiniteArithmetic;

  Float num1 ("203212.231");
  Float num2 ("42132.0322");

  LOG(num1.Multiply(num2));
  // Output = 8561744259.9258382
.
```

5.2.10 MultiplyByDigit

`Float MultiplyByDigit(int)`

This function is called internally by `Divide`. It multiplies an `Float` by a single digit`(0-9)`.

```

  using namespace InfiniteArithmetic;

  Float num1 ("203212.231");

  LOG(num1.MultiplyByDigit(2));
  // Output = 406424.462
.
```

5.2.11 Divide

`Float Divide(Float)`

The function `Divide` internally does human long division. The `/` operator has been overloaded for division. Here's an example:

```

  using namespace InfiniteArithmetic;

  Float num1 ("203212.231");
  Float num2 ("42132.0322");

  LOG(num1.Divide(num2));
  // Output = 4.8232240504173923991257179377167569904211 7...
.
```

5.2.12 SetPrecision

`void SetPrecision(int64_t)`

This sets the number of digits after decimal that should be calculated while division happens.

5.2.13 Compare

`int16_t Compare(Float)`

The `Compare` function returns a value in `{-1, 0, 1}` depending on which number is greater or lesser. All the comparison operators have been overloaded to do the same.

```

  using namespace InfiniteArithmetic;

  Float num1 ("2.2");
  Float num2 ("5.5");

  LOG(num1.Compare(num2));
  // Output = -1
.
```

5.2.14 Complement

`Float Complement() const`

The `Complement` function takes the complement of a number with respect to 9. For example, 102's complement is 898.
It is a private member function.

5.2.15 Negate

`Float Negate()`

The `Negate` function changes the sign of the number. The unary `-` operator also does the same.

```

  using namespace InfiniteArithmetic;

  Float num1 ("5.5");
  LOG(num1.Negate());
  // Output = -5.5
.
```

5.2.16 isZero

`bool isZero()`

The `isZero` function checks if the number is 0.

```

  using namespace InfiniteArithmetic;

  Float num1 ("0.0");
  LOG(num1.isZero());
  // Output = 1

  Float num2 ("2.1");
  LOG(num2.isZero());
  // Output = 0
.
```

5.2.17 MatchDigits

`void MatchDigits(Float &, Float &)`

The `MatchDigits` method is a static function that makes the vectors of the two numbers to equal length. This is a private function.

5.2.18 ResizeEnds

`void ResizeEnds(Float &, Float &)`

This is similar to the `MatchDigits` function but it works for the fractional part of the floating point number.

5.2.19 VerifyString

`void VerifyString(std::string)`

The `VerifyString` checks for any non-digit characters and raises an error if found. This is also a private function.

5.2.20 PopZero

`void PopZero()`

This function removes any redundant zeroes present in the number.

6 Conclusion

6.1 Shortcomings

1. Some of the algorithms are really slow. If your program has many operations in it, then it becomes ineffiecient.
2. Each float Division takes nearly 1.5s.
3. Most operations are written with pass-by-copy.

6.2 Verification

The code has been verified using python scripts.

6.3 Learnings

1. Version Control Using Git.
2. Library creation and usage.
3. Efficient Debugging.