

---

# Infinite Precision Library Documentation

Sathvik Reddy Bhavanam

CS23BTECH11056

April 21, 2024

# Contents

<b>1</b>	<b>Installation</b>	<b>4</b>
1.1	Cloning The Repository . . . . .	4
1.2	Building The Executable . . . . .	4
1.3	Using The Executable . . . . .	4
1.4	Building The Library . . . . .	4
1.5	Linking The Library . . . . .	4
<b>2</b>	<b>Design</b>	<b>5</b>
2.1	Underlying Implementation . . . . .	5
2.2	Limitations Of Vector . . . . .	5
2.3	Error Handling . . . . .	5
2.4	Memory Management . . . . .	5
<b>3</b>	<b>API Tables</b>	<b>6</b>
<b>4</b>	<b>Integer Library</b>	<b>7</b>
4.1	Introduction . . . . .	7
4.2	API Reference . . . . .	7
4.2.1	Constructors . . . . .	7
4.2.2	Destructor . . . . .	7
4.2.3	parse . . . . .	7
4.2.4	Assign . . . . .	8
4.2.5	Print . . . . .	8
4.2.6	Input . . . . .	8
4.2.7	Add . . . . .	8
4.2.8	Add2 . . . . .	9
4.2.9	Subtract . . . . .	9
4.2.10	Multiply . . . . .	9
4.2.11	MultiplyByDigit . . . . .	9
4.2.12	Divide . . . . .	10
4.2.13	Mod . . . . .	10
4.2.14	Compare . . . . .	10
4.2.15	Complement . . . . .	10
4.2.16	Negate . . . . .	11
4.2.17	isZero . . . . .	11
4.2.18	MatchDigits . . . . .	11
4.2.19	VerifyString . . . . .	11
4.2.20	PopZero . . . . .	11
<b>5</b>	<b>Float Library</b>	<b>12</b>
5.1	Introduction . . . . .	12
5.2	API Reference . . . . .	12
5.2.1	Constructors . . . . .	12
5.2.2	Destructor . . . . .	12
5.2.3	parse . . . . .	12
5.2.4	Assign . . . . .	13
5.2.5	Print . . . . .	13
5.2.6	Input . . . . .	13
5.2.7	Add . . . . .	13
5.2.8	Subtract . . . . .	14
5.2.9	Multiply . . . . .	14
5.2.10	MultiplyByDigit . . . . .	14
5.2.11	Divide . . . . .	14
5.2.12	Mod . . . . .	15

5.2.13	Compare . . . . .	15
5.2.14	Complement . . . . .	15
5.2.15	Negate . . . . .	15
5.2.16	isZero . . . . .	16
5.2.17	MatchDigits . . . . .	16
5.2.18	ResizeEnds . . . . .	16
5.2.19	VerifyString . . . . .	16
5.2.20	PopZero . . . . .	16
<b>6</b>	<b>Conclusion</b>	<b>17</b>

# 1 Installation

This section consists 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 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 inconvenient 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.

### 3 API Tables

Functions	Purpose
<code>Assign</code>	Used to assign the object to another value
<code>Add</code>	Adds two numbers together
<code>Subtract</code>	Subtracts one number from the other
<code>Multiply</code>	Multiplies two numbers
<code>Divide</code>	Divides two numbers
<code>Negate</code>	Negates the number
<code>isZero</code>	Tells if the number is zero or not
<code>parse</code>	Returns an instance of the Float class
<code>@Complement</code>	Takes the complement of given number
<code>@MatchDigits</code>	Matches the digits of two numbers such that they are equal
<code>@PopZero</code>	Removes redundant zeroes from the number
<code>@Zero</code>	Displays the number

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

## 4 Integer Library

### 4.1 Introduction

In this section, describe the purpose and functionalities of your Integer library. Explain how it provides infinite precision arithmetic for integers.

### 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`.

```
1 #include "Integer.h"
2 #include "Float.h"
3
4 int main()
5 {
6     LOG(InfiniteArithmetic::Integer());
7     // Output = 0
8 }
```

##### `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.

```
1
2     LOG(InfiniteArithmetic::Integer("212"));
3     // Output = 212
4 .
```

##### `Integer(const Integer &obj)`

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

```
1
2     namespace InfiniteArithmetic = IA;
3     LOG(IA::Integer(IA::Integer("110")));
4     // Output = 110
5 .
```

#### 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.

```
1
2  IA::Integer num1 ("102");
3  IA::Integer num2;
4  num2.Assign(num1);
5  LOG(num2);
6  // Output = 102
7  .
```

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.

```
1
2  using namespace InfiniteArithmetic;
3
4  Integer num1("322");
5
6  std::cout << num1 << std::endl;
7  // Output = 322
8  .
```

#### 4.2.6 Input

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

```
1
2  using namespace InfiniteArithmetic;
3
4  Integer num1;
5  Integer num2;
6
7  std::cin >> num1 >> num2;
8  .
```

#### 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.

```
1
2  using namespace InfiniteArithmetic;
3
4  Integer num1 ("1023");
5  Integer num2 ("3213");
6
7  LOG(num1.Add(num2));
8  // Output = 4236
9  .
```



### 4.2.8 Add2

#### Integer Add2(Integer &)

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

```
1
2 using namespace InfiniteArithmetic;
3
4 Integer num1 ("1023");
5 Integer num2 ("3213");
6
7 LOG(num1.Add2(num2));
8 // Output = 4236
9 .
```

### 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.

```
1 using namespace InfiniteArithmetic;
2
3 Integer num1 ("1023");
4 Integer num2 ("3213");
5
6 LOG(num1.Subtract(num2));
7 // Output = -2190
8 .
```

### 4.2.10 Multiply

#### Integer Multiply(Integer)

The \* operator has been overloaded for multiplication.

```
1 using namespace InfiniteArithmetic;
2
3 Integer num1 ("322");
4 Integer num2 ("4221");
5
6 LOG(num1.Multiply(num2));
7 // Output = 1359162
8 .
```

### 4.2.11 MultiplyByDigit

#### Integer MultiplyByDigit(int)

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

```
1
2 using namespace InfiniteArithmetic;
3
4 Integer num1 ("322");
5 LOG(num1.MultiplyByDigit(2));
6 // Output = 644
7 .
```

#### 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

Describe the purpose and functionalities of your Float library here. Explain how it provides infinite precision arithmetic for floating-point numbers.

### 5.2 API Reference

Document the functions and methods provided by the Float library, similar to the Float library section.

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`.

```
1 #include "Float.h"
2 #include "Float.h"
3
4 int main()
5 {
6     LOG(InfiniteArithmetic::Float());
7     // Output = 0
8 }
```

##### `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.

```
1
2     LOG(InfiniteArithmetic::Float("212"));
3     // Output = 212
4 .
```

##### `Float(const Float &obj)`

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

```
1
2     namespace InfiniteArithmetic = IA;
3     LOG(IA::Float(IA::Float("110")));
4     // Output = 110
5 .
```

#### 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.

```
1
2   IA::Float num1 ("102");
3   IA::Float num2;
4   num2.Assign(num1);
5   LOG(num2);
6   // Output = 102
7 .
```

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.

```
1
2   using namespace InfiniteArithmetic;
3
4   Float num1("322");
5
6   std::cout << num1 << std::endl;
7   // Output = 322
8 .
```

### 5.2.6 Input

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

```
1
2   using namespace InfiniteArithmetic;
3
4   Float num1;
5   Float num2;
6
7   std::cin >> num1 >> num2;
8 .
```

### 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.

```
1
2   using namespace InfiniteArithmetic;
3
4   Float num1 ("1023");
5   Float num2 ("3213");
6
7   LOG(num1.Add(num2));
8   // Output = 4236
9 .
```

### 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.

```
1 using namespace InfiniteArithmetic;
2
3 Float num1 ("1023");
4 Float num2 ("3213");
5
6 LOG(num1.Subtract(num2));
7 // Output = -2190
8 .
```

### 5.2.9 Multiply

#### Float Multiply(Float)

The `*` operator has been overloaded for multiplication.

```
1 using namespace InfiniteArithmetic;
2
3 Float num1 ("322");
4 Float num2 ("4221");
5
6 LOG(num1.Multiply(num2));
7 // Output = 1359162
8 .
```

### 5.2.10 MultiplyByDigit

#### Float MultiplyByDigit(int)

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

```
1 using namespace InfiniteArithmetic;
2
3
4 Float num1 ("322");
5 LOG(num1.MultiplyByDigit(2));
6 // Output = 644
7 .
```

### 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:

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

### 5.2.12 Mod

#### Float Mod(Float)

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

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

### 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.

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

### 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.

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

### 5.2.16 isZero

`bool isZero()`

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

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

### 5.2.17 MatchDigits

`void MatchDigits(Float &, Float &)`

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

### 5.2.18 ResizeEnds

/Sample Text/

### 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