Project #1 – BigInt

In this project you are asked to implement a `BigInt` class capable of representing integers of any size. The following sample code demonstrates some of the capabilities of this class:

```
BigInt bi1("67190236716071023409176961234786090001");
BigInt bi2("-16719238679867620194861264901235");
BigInt bi3("17767012093706970293750297356096723069722");

// Outputs:
// -17699822024183286069017322343474585992006
cout << bi1 + bi2 - bi3;
```

Requirements

- `BigInt` must support the following interface:

```
namespace Project1
{
	class BigInt
	{
		friend const BigInt operator+(const BigInt &, const BigInt &);
		friend const BigInt operator-(const BigInt &, const BigInt &);
		friend bool operator==(const BigInt &, const BigInt &);
		friend bool operator!=(const BigInt &, const BigInt &);
		friend bool operator<(const BigInt &, const BigInt &);
		friend bool operator<=(const BigInt &, const BigInt &);
		friend bool operator>(const BigInt &, const BigInt &);
		friend bool operator>=(const BigInt &, const BigInt &);
		friend ostream &operator<<(ostream &, const BigInt &);
		friend istream &operator>>(istream &, BigInt &);

	public:
		BigInt();
		BigInt(const BigInt &);
		BigInt(long long);
		// Throws invalid_argument if string malformed (contains
		// anything other than a legally formatted number)
		BigInt(const string &);

		const BigInt &operator=(const BigInt &);
		const BigInt &operator+=(const BigInt &);
		const BigInt &operator-=(const BigInt &);

	private:
		// You decide what goes here...
	}
}
```

- Negative `BigInt`s must be supported
- All functions must be strongly exception safe & exception neutral
- Algorithms should be preferred over `for`/`do`/`while` loops

- Related operations (e.g. `operator+` and `operator+=`) must be properly implemented in terms of each other
- Duplicate code must refactored into helper functions
- You may implement the internals of `BigInt` however you wish; the only restriction is that you may not use third-party big integer libraries (though you may look at them for inspiration)
- Your code must be ANSI-compliant, well organized, & stylistically clean

Grading - (150 total points available)

1. **(110 points)** One point for each passing unit test (there are 110 unit tests provided). Your `BigInt` implementation will be run against these tests.
2. **(10 points)** BigInt is strongly exception safe.
3. **(10 points)** BigInt is exception neutral.
4. **(10 points)** BigInt has no memory leaks.
5. **(10 points)** BigInt is implemented using only ANSI-compliant C++ features, the code is clean (e.g. no duplicate code), the code uses best practices (e.g. operator+ implemented in terms of operator+=).

Turning in the assignment

- Place your BigInt.h and BigInt.cpp files in a zip file. Submit this zip file.