BigInt

Arbitrary-sized integer class for C++

Contents

Highlights

- No additional dependencies apart from the standard library.
- Modern C++ (compiles with C++11 / C++14 / C++17).
- No special compiling or linking required.

Usage

1. Include the header file:

   ```
   #include "BigInt.hpp"   // the actual path may vary
   ```

2. Create objects of the `BigInt` class, and do what you got to do!

   ```
   BigInt big1 = 1234567890, big2;
   big2 = "9876543210123456789098765432101234567890";

   std::cout << big1 * big2 * 123456 << "\n";
   // Output: 1505331490682966620443288524512589666204282352096057600
   ```

Features

Operators

- **Assignment: `=`**

  The second operand can either be a `BigInt`, an integer (up to `long long`) or a string (`std::string` or a string literal).

  ```c++
  big1 = 1234567890;
  big1 = "123456789012345678901234567890";
  big1 = big2;
  ```

~~~
* #### Unary arithmetic: `+`, `-`
  ```c++
  big1 = +big2;   // doesn't return the absolute value
  big1 = -big2;
~~~

- **Binary arithmetic: `+`, `-`, `*`, `/`, `%`**

  One of the operands has to be a `BigInt` and the other can be a `BigInt`, an integer (up to `long long`) or a string (`std::string` or a string literal).

  ```c++
  big1 = big2 + 1234567890;
  big1 = big2 - "123456789012345678901234567890";
  big1 = big2 * big3;
  big1 = 1234567890 / big2;
  big1 = "123456789012345678901234567890" % big2;
  ```

~~~
* #### Arithmetic-assignment: `+=`, `-=`, `*=`, `/=`, `%=`
  The second operand can either be a `BigInt`, an integer (up to `long long`)
  or a string (`std::string` or a string literal).

  ```c++
  big1 += big2;
  big1 -= 1234567890;
  big1 *= "123456789012345678901234567890";
  big1 /= big2;
  big1 %= 1234567890;
~~~

- **Increment and decrement: `++`, `--`**

```
big1 = ++big2;    // pre-increment
big1 = --big2;    // pre-decrement

big1 = big2++;    // post-increment
big1 = big2--;    // post-decrement
```

- **Relational: `<`, `>`, `<=`, `>=`, `==`, `!=`**

  One of the operands has to be a `BigInt` and the other can be a `BigInt`, an integer (up to `long long`) or a string (`std::string` or a string literal).

  ```
  if (big1 < 1234567890
      || big1 > "123456789012345678901234567890"
      || big1 <= big2
      || 1234567890 >= big1
      || "123456789012345678901234567890" == big1
      || big1 != big3) {
      ...
  }
  ```

- **I/O stream: `<<`, `>>`**

  ```
  std::cout << big1 << ", " << big2 << "\n";
  output_file << big1 << ", " << big2 << "\n";

  std::cin >> big1 >> big2;
  input_file >> big1 >> big2;
  ```

Functions

- **Conversion: `to_string`, `to_int`, `to_long`, `to_long_long`**

  Convert a `BigInt` to either a `string`, `int`, `long`, or `long long`.

  Note: If the `BigInt` is beyond the range of the target type, an [out_of_range exception][out_of_range-exception] is thrown.

  ```
  some_str = big1.to_string();

  some_int = big1.to_int();

  some_long = big1.to_long();

  some_long_long = big1.to_long_long();
  ```

- **Math**
  - **`abs`**

    Get the absolute value of a `BigInt`.

    ```
    big1 = abs(big2);
    ```

  - **`big_pow10`**

    Get a `BigInt` equal to *10*^{exp}.

    ```
    big1 = big_pow10(5000);    // big1 = 10^5000
    ```

  - **`gcd`**

    Get the greatest common divisor (GCD aka. HCF) of two `BigInt`s. One of the arguments can be an integer (up to `long long`) or a string (`std::string` or a string literal).

    ```
    big1 = gcd(big2, big3);
    big1 = gcd(big2, 1234567890);
    big1 = gcd(big2, "123456789012345678901234567890");
    big1 = gcd(1234567890, big2);
    big1 = gcd("123456789012345678901234567890", big2);
    ```

  - **`lcm`**

    Get the least common multiple (LCM) of two `BigInt`s. One of the arguments can be an integer (up to `long long`) or a string (`std::string` or a string literal).

    ```
    big1 = lcm(big2, big3);
    big1 = lcm(big2, 1234567890);
    big1 = lcm(big2, "123456789012345678901234567890");
    big1 = lcm(1234567890, big2);
    big1 = lcm("123456789012345678901234567890", big2);
    ```

  - **`pow`**

    Get the value of *base*^{exp} as a `BigInt`. The base can either be a `BigInt`, an integer (up to `long long`) or a string (`std::string` or a string literal).

```
big1 = pow(big2, 789);
big1 = pow(987654321LL, 456);    // suffix literal with LL to prevent
conflicts
big1 = pow("1234567890", 123);
```

  - `sqrt`

    Get the integer square root of a `BigInt`.

    ```
    big1 = sqrt(big2);
    ```

- **Random**

  - `big_random`

    Get a random `BigInt`, that either has a random number of digits (up to 1000), or a specific number of digits.

    ```
    // get a random BigInt that has a random number of digits (up to 1000):
    big1 = big_random();

    // get a random BigInt that has 12345 digits:
    big1 = big_random(12345);
    ```