有理数

我们知道，当前计算机中常用的浮点数表示方式是离散的，本题需要你实现一个以分数形式表示的有理数。

你需要实现如下的功能：

- 用分子（numerator）和分母（denominator）构造一个有理数
  - 如果分母为 0，抛出 `std::logic_error` 异常，异常信息为 `denominator must be != 0`
- 拷贝构造函数
- 重载的 `operator=`
- 有理数的加、减、乘、除
  - 保证不会出现相减结果为负的情况
  - 保证不会出现除数为 0 的情况
- 获取有理数的分子和分母
- 向 `std::ostream` 对象输出一个有理数，即实现 `operator<<` 的重载
  - 如果分子为 0，则不论分母的值为多少，都输出 `0`
  - 将分子记为 `N`、分母记为 `D`，分子和分母的最大公约数（GCD）为 `G`，记 `N / G` 的值为 `M`、`D / G` 的值为 `E`，则输出为 `M/E`，注意**不包含**空格！
  - 如果约分后的分母为 1，则不输出分母
  - 示例
    - `std::cout << Rational<int>(3, 4)`，得到 `3/4`
    - `std::cout << Rational<int>(2, 4)`，得到 `1/2`
    - `std::cout << Rational<int>(4, 2)`，得到 `2`
- 实现 `_r` 后缀以支持通过 `"3/4"_r` 的形式构建一个有理数
  - 该特性称为“User-Defined Literals”，从 C++11 引入
  - 参见官方文档、这篇文章和这篇文章

输入、输出描述

本题不需要处理输入输出。

输入格式参见代码框架的 `main` 函数，输出格式参见代码框架的 `test_*` 系列函数。

示例

见代码框架中的 `test_*` 系列函数。

扩展

- 假设我们用 `int` 作为 `Rational` 的模板参数。如果想要获取 `double` 类型的分子或分母，可以通过先获取到 `int` 值然后进行强制类型转换。然而，还有另一种方式：通过模板参数指

定返回值的类型。探索此种实现方式，了解 type traits 的基本概念和标准库提供的 `is_convertible` traits

```
// 示例：通过模板参数指定返回值类型
Rational<int> r(3, 4);
double n = r.numerator<double>();
```

- 如何确保模板参数 `T` 对应的类型一定是可以进行算术运算的？了解相关的 type traits

代码框架

```
#include <cctype>
#include <cstring>
#include <functional>
#include <iostream>
#include <stdexcept>
#include <string>
#include <unordered_map>

template <typename T>
class Rational {
 public:
  Rational(const T &n, const T &d);
  Rational(const Rational<T> &rhs);

  T numerator() const;
  T denominator() const;

  Rational<T> operator+(const Rational<T> &rhs) const;
  Rational<T> operator-(const Rational<T> &rhs) const;
  Rational<T> operator*(const Rational<T> &rhs) const;
  Rational<T> operator/(const Rational<T> &rhs) const;

  Rational<T> &operator=(const Rational<T> &rhs);

  friend std::ostream &operator<<(std::ostream &out, const Rational<T>
&r);

  // TODO: your code
};

Rational<int> operator""_r(const char *str, size_t len);

void test_1() {
  Rational<int> r(3, 4);
```

```
  std::cout << r << std::endl;
}

void test_2() {
  bool exception_thrown = false;
  bool expected_message = false;
  try {
    Rational<int> r = Rational<int>(1, 0);
  } catch (std::logic_error &exn) {
    exception_thrown = true;
    if (!strncmp(exn.what(), "denominator must be != 0", 24)) {
      expected_message = true;
    }
  } catch (...) {
  }
  if (exception_thrown) {
    std::cout << "std::logic_error thrown!" << std::endl;
    if (expected_message) {
      std::cout << "the message is as expected." << std::endl;
    }
  } else {
    std::cout << "Oops!" << std::endl;
  }
}

void test_3() {
  Rational<int> r(3, 4);
  std::cout << r.numerator() << ' ' << r.denominator() << std::endl;
}

void test_4() {
  Rational<int> lhs(1, 6), rhs(1, 3);
  std::cout << (lhs + rhs) << std::endl;
}

void test_5() {
  Rational<int> lhs(1, 2), rhs(1, 6);
  std::cout << (lhs - rhs) << std::endl;
}

void test_6() {
  Rational<int> lhs(2, 4), rhs(4, 6);
  std::cout << (lhs * rhs) << std::endl;
}

void test_7() {
  Rational<int> lhs(2, 4), rhs(4, 6);
  std::cout << (lhs / rhs) << std::endl;
}
```

```
void test_8() {
  Rational<int> r(3, 4);
  std::cout << r << std::endl;
  Rational<int> rhs(101, 203);
  r = rhs;
  std::cout << r << ' ' << rhs << std::endl;
}

void test_9() {
  auto r = "3/4"_r;
  std::cout << r << std::endl;
}

void test_10() { std::cout << Rational<int>(4, 2) << std::endl; }

void test_11() {
  std::cout << (Rational<int>(1, 2) - Rational<int>(2, 4)) <<
std::endl;
}

void test_12() { std::cout << Rational<int>(3, 6) << std::endl; }

int main() {
  std::unordered_map<std::string, std::function<void()>>
test_cases_by_name = {
      {"test_1", test_1},   {"test_2", test_2},   {"test_3", test_3},
      {"test_4", test_4},   {"test_5", test_5},   {"test_6", test_6},
      {"test_7", test_7},   {"test_8", test_8},   {"test_9", test_9},
      {"test_10", test_10}, {"test_11", test_11}, {"test_12", test_12},
  };
  std::string tname;
  std::cin >> tname;
  auto it = test_cases_by_name.find(tname);
  if (it == test_cases_by_name.end()) {
    std::cout << "输入只能是 test_<N>, 其中 <N> 可取整数 1 到 12." <<
std::endl;
    return 1;
  }
  (it->second)();
}
```