Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ФУНДАМЕНТАЛЬНЫЕ НАУКИ»

КАФЕДРА «ПРИКЛАДНАЯ МАТЕМАТИКА»

Лабораторная работа № 7

по дисциплине «Типы и структуры данных»

Тема Долгая арифметика

Студент Лямин И.С.

Группа ФН12-31Б

Преподаватели Волкова Л.Л.

Москва, 2024

Содержание

ВВЕДЕНИЕ **3**

1 Аналитическая часть **4**

1.1 Система счислений в виде простых множителей 4

1.2 Длинная арифметика 4

2 Конструкторская часть **5**

2.1 Структура хранения длинных чисел 5

2.2 Алгоритм преобразования числа в систему множителей 5

2.3 Алгоритм сложения/вычитания чисел в системе множителей 6

2.4 Алгоритм умножения чисел в системе множителей 8

2.5 Алгоритм получения целой части и остатка при делении чисел в системе множителей 8

2.6 Алгоритм сравнения чисел в системе множителей 10

3 Технологическая часть **12**

3.1 Выбор средств реализации 12

3.2 Реализация алгоритмов 12

3.3 Тестирование программы 12

ЗАКЛЮЧЕНИЕ **15**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **16**

Приложение А **17**

ВВЕДЕНИЕ

Цель работы — освоение принципов длинной арифметики.

Для достижения цели необходимо выполнить следующие задачи:

1) описать принципы переведения чисел из десятичной системы счислений в систему состоящую из множителей числа и обратно,
2) описать принципы операций в данной системе счислений,
3) реализовать операции над длинными числами,
4) протестировать программу.

1 Аналитическая часть

1.1 Система счислений в виде простых множителей

Система счисления в виде множителей числа — это способ представления чисел в виде произведения их простых множителей. Такой подход используется, например, в задачах, связанных с теорией чисел, криптографией или для специальных арифметических операций. Любое число представимо в виде произведения своих простых множителей, при этом такое представление единственно с точностью до перестановки множителей.

$$N = p_1^{e_1} \cdot p_2^{e_2} \cdot ... \cdot p_n^{e_n} \tag{1.1}$$

Где $p_1, p_2, ..., p_n$ – простые множители числа, а $e_1, e_2, ..., e_n$ – степени множителей. Операции в такой системе:

— умножение – степени всех делителей суммируются,

— деление – степени всех делителей вычитаются,

— сложение/вычитание – при этих операция, сначала приводим общие множители(делители) и получаем число равное их произведению, множители обоих числе не попавших в общие перемножаем, множители первого перемножаем между собой и множители второго тоже между собой, и проводим соответствующую операцию(сложение/вычитание), полученное в результате этого действия число представляем в виде множителей и умножаем на число полученное от произведения всех общих множителей.

1.2 Длинная арифметика

Длинная арифметика – это набор программных средств, позволяющий работать с числами гораздо больших величин, чем это позволяют стандартные типы данных. Операции в длинной арифметике выполняются поразрядно.

2 Конструкторская часть

2.1 Структура хранения длинных чисел

Длинное число в программе представляется в виде структуры. Структура `Number_in_factors` описывает число в системе счисления основанной на множителях. Она включает в себя следующие поля:

1) `map<int, int> number` – хранит в себе все множители числа и их степени,

2) `int negative` – хранит знак числа,

3) `bool empty` – если число равно нулю, то переменная истинна, в обратном случае она ложна.

Для перевода числа из десятичной системы в систему множителей реализован метод `convert_to_factors` структуры `Number_in_factors`, которая на вход получает строку `inp` И работает по следующему алгоритму.

2.2 Алгоритм преобразования числа в систему множителей

Для преобразования числа, представленного в виде строки, в систему множителей используется функция `convert_to_factors`, которая на вход получает строку `inp`. Алгоритм, реализованный в этой функции, состоит из следующих шагов:

Поля и начальная настройка

1) `num_10`
   - тип: `unsigned long long`,
   - описание: число, полученное из строки в десятичной системе счисления,
   - начальное значение: `0`.
2) `digit`
   - тип: `int`,
   - описание: число соответствующее разряду числа в десятичной системе,
   - начальное значение: `inp.size() - 1`.
3) `empty`
   - тип: `bool`,
   - описание: флаг для обозначения нулевого числа,
   - начальное значение: `1`.
4) `number`
   - тип: `map<int, int>`,
   - описание: хранилище степени каждого простого множителя,
   - начальное значение: 1, 1.

Этапы работы алгоритма

1) **Преобразование строки в десятичное число (**`num_10`**):**
   — для каждой цифры строки `inp` выполняется:
      – если текущий символ `'-'`, проверяем флаг отрицательности. Если отрицательность уже установлена, выводим сообщение об ошибке,
      – если текущий символ `'0'`, пропускаем его и уменьшаем `digit`,
      – в остальных случаях преобразуем символ в число, умножаем его на 10^{digit}, прибавляем к `num_10` и уменьшаем `digit`.
2) **Обработка чисел, равных нулю:**
   — если `num_10 == 0`, устанавливаем `empty = 1` и очищаем `number`,
   — завершаем работу функции.
3) **Разложение числа на простые множители:**
   — используем вектор `divs`, содержащий список простых чисел, полученный в результате работы функции `FIND_DIVS`, находящей все простые числа до числа до 2^{22}, по алгоритму `"решето Эратосфена"`,
   — для каждого простого числа в `divs`:
      – пока `num_10 % divs[loc_ind] == 0`, увеличиваем степень этого множителя в `number` и делим `num_10`.
   — если число не разложено полностью, используем перебор оставшихся чисел.
4) **Обработка ошибок:**
   — если `loc_ind >= divs.size()`, выводим сообщение о том, что искомые делители очень большие(больше, чем 2^{22}) и их вычисление может занять много времени и начинаем перебор оставшихся делителей,
   — если `num_10` == 1, то заканчиваем алгоритм.

2.3 Алгоритм сложения/вычитания чисел в системе множителей

Для сложения чисел, представленных в виде системы множителей, используется перегрузка оператора `+`. Функция на вход принимает объект `_2` типа `Number_in_factors`, представляющий второе слагаемое. Алгоритм, реализованный в функции, включает следующие шаги:

Поля и начальная настройка

1) `common_factors`
   — тип: `Number_in_factors`,
   — описание: хранит общие множители двух чисел,
   — начальное значение: единичный объект `Number_in_factors`.
2) `small_sum_1, small_sum_2`

— тип: `long long`,

— описание: значения чисел в десятичной системе без общих множителей,

— начальное значение: 0 для пустых чисел, ± 1 для ненулевых (с учётом знака).

3) `small_sum`

— тип: `Number_in_factors`,

— описание: итоговая сумма чисел в виде множителей,

— начальное значение: единичный объект `Number_in_factors`.

Этапы работы алгоритма

1) **Поиск общих множителей и преобразование чисел в десятичные значения:**

— для каждого множителя второго числа (`_2.number`):

– если множитель является общим для чисел:

* добавляем его минимальную степень в `common_factors`.

* остаток степени добавляем к `small_sum_1` или `small_sum_2`, в зависимости от числа, в котором этот остаток присутствует.

– если множитель отсутствует в первом числе:

* умножаем `small_sum_2` на значение множителя в его степени.

2) **обработка уникальных множителей первого числа:**

— для каждого множителя первого числа (`this->number`), который отсутствует во втором числе:

– умножаем `small_sum_1` на значение множителя в его степени.

3) **сумма промежуточных значений и преобразование результата в систему множителей:**

— вычисляем сумму `small_sum_1 + small_sum_2`,

— если сумма равна 0, устанавливаем флаг `empty` в `small_sum` и функция возвращает `small_sum`,

— если сумма отрицательная, устанавливаем `negative = -1`,

— преобразуем абсолютное значение суммы в систему множителей, используя метод `convert_to_factors`.

4) **возврат результата:**

— возвращаем произведение `common_factors` и `small_sum`.

Алгоритм находящий разность чисел так же реализован в методе, описывающем поведение переопределённого оператора `'-'`. И отличается от алгоритма суммы, только тем, что на каждом шаге ищется разность, а на последнем шаге алгоритма, перед применением метода `convert_to_factors`, определяем модуль разности чисел `small_sum_1` и `small_sum_2`, и передаём его в метод `convert_to_factors` присваивая полю `negative` новой структуры значение соответствующее знаку разности чисел `small_sum_1` и `small_sum_2`.

2.4 Алгоритм умножения чисел в системе множителей

Для умножения чисел, представленных в виде системы множителей, используется перегрузка оператора `*`. Функция на вход принимает объект `_2` типа `Number_in_factors`, представляющий второй множитель. Алгоритм, реализованный в функции, включает следующие шаги:

Поля и начальная настройка

1) `res`
   - тип: `Number_in_factors`,
   - описание: хранит значение произведения двух чисел,
   - начальное значение полей:
     - `negative = this->negative * _2.negative`,
     - `number = this->number`
     - `empty = 0`

Этапы работы алгоритма

1) **Поиск всех множителей обоих чисел:**
   - для каждого множителя второго числа (`_2.number`):
     - если такого множителя ещё нет в `res.number`, то добавляем его,
     - иначе степень этого множителя в числе `_2` прибавляем к значению `res.number[множитель]` множитель отсутствует в первом числе.
2) **Проверка умножения на ноль:**
   - если значение одного из флагов `this->empty` или `_2.empty` равно `true`:
     - присваиваем флагу `res.empty` значение true и чистим словарь `res.number`.

2.5 Алгоритм получения целой части и остатка при делении чисел в системе множителей

Для данной задачи, реализована функция `div`. Она принимает на вход три аргумента: указатели на объекты `Number_in_factors` (`_1` и `_2`) и строку `mode`, которая определяет, будет ли выполнено целочисленное деление или операция взятия остатка от деления. Алгоритм, реализованный в функции, включает следующие шаги:

Поля и начальная настройка

1) `sign`
   - тип: `int`,
   - описание: Переменная для хранения знака результата операции деления, вы-

числяется как произведение знаков `_1.negative` и `_2.negative`.

2) `frac`

— тип: `vector<unsigned long long>`,

— описание: вектор, хранящий числовые значения,

— начальное значение: результат функции `cancellation`, которая возвращает два максимально сокращённых(без общих делителей) числа.

3) `res`

— тип: `Number_in_factors`,

— описание: результат деления двух чисел в системе множителей, представленный также в виде системы множителей,

— начальное значение: единичный объект типа `Number_in_factors`.

4) `a`

— тип: `string`,

— описание: строка, которая будет содержать строковое представление числа, полученного в результате деления или операции взятия остатка,

— начальное значение: пустая строка.

Этапы работы алгоритма

1) **Проверка на пустое число:**

— проверяется, является ли второе число (`_2`) пустым. Если да, то результат устанавливается как пустое число, и алгоритм выводит ошибку деления.

2) **Приведение чисел к простым множителям:**

— используется функция `cancellation`, чтобы преобразовать оба числа (`_1` и `_2`), сохраняем их в векторе `frac`,

— вектор `frac` после выполнения функции `cancellation` содержит два числа: числитель и знаменатель в системе множителей.

3) **Определение операции (деление или остаток):**

— если режим `mode` равен `"div"`, выполняется целочисленное деление числителя на знаменатель,

— если режим `mode` равен `"mod"`, выполняется взятие остатка от деления числителя на знаменатель.

4) **Конвертация результата в систему множителей:**

— результат операции деления или взятия остатка (с учетом знака) преобразуется в строку и передается в метод `convert_to_factors` для преобразования в систему множителей.

5) **Возврат результата:**

— результат операции деления или взятия остатка в виде системы множителей возвращается из функции.

2.6 Алгоритм сравнения чисел в системе множителей

Для данной задачи реализована функция `compare`, которая принимает два аргумента: объекты `Number_in_factors` (`_1` и `_2`) и возвращает результат сравнения этих чисел. Алгоритм, реализованный в функции, включает следующие шаги:

Поля и начальная настройка

1) `res`
   — тип: `vector<Number_in_factors>`,
   — описание: вектор, хранящий два объекта типа `Number_in_factors`, который используется для хранения промежуточных результатов при сравнении чисел,
   — начальное значение: пустой вектор из двух объектов типа `Number_in_factors`.
2) `range`
   — тип: `int`,
   — описание: переменная, используемая для хранения разницы между степенями одинаковых простых множителей двух чисел.
   — начальное значение: 0.
3) `lg_1, lg_2`
   — тип: `unsigned long`,
   — описание: переменные для хранения логарифмических значений чисел `_1` и `_2`, которые используются для более точного сравнения чисел с большими значениями,
   — начальное значение: 0,

Этапы работы алгоритма

1) **Проверка на пустое число:**
   — проверяется, является ли одно из чисел пустым. Если одно из чисел пустое (`_2.empty == 1`), то алгоритм сразу возвращает `-1` или `1`, в зависимости от знака числа `_1`,
   — если оба числа пустые, то возвращается `0`, так как числа равны.
2) **Проверка на знак чисел:**
   — если одно число отрицательное, а другое положительное, то возвращается `-1` или `1` в зависимости от знаков чисел,
   — если оба числа имеют одинаковые знаки, продолжаем дальнейшее сравнение.
3) **Сравнение чисел на основе их простых множителей:**
   — для каждого простого множителя второго числа (`_2.number`) проверяется его наличие в первом числе (`_1.number`),
      – если множитель отсутствует в первом числе, то его степень добавляется

в результат во второй объект `res[1]`,

– если множитель присутствует в обоих числах, то вычисляется разница степеней. Если разница положительная, то множитель добавляется в первый объект `res[0]`, если отрицательная — в `res[1]`.

— после обработки множителей второго числа, остаток множителей из первого числа (`_1.number`) добавляется в первый объект `res[0]`.

4) **Логарифмическое сравнение чисел:**

— для каждого множителя в `res[0].number` и `res[1].number` вычисляются их логарифмы по основанию `e`,

— логарифмы чисел `_1` и `_2` сравниваются между собой для того, чтобы определить, какое число больше.

5) **Возврат результата:**

— если логарифм первого числа больше, возвращается `1`, если меньше, то `-1`, если они равны, то возвращается `0`.

3 Технологическая часть

3.1 Выбор средств реализации

Для программной реализации алгоритма использовалась среда разработки Visual Studio 2022, язык программирования, на котором была выполнена реализации алгоритмов — C++. Для компиляции кода использовался компилятор MSVC. Исследование проводилось на ноутбуке (64–разрядная операционная система, процессор x64, частота процессора 3.1 ГГц, модель процессора 12th Gen Intel(R) Core(TM) i5-12500H, оперативная память 16 ГБ)

3.2 Реализация алгоритмов

В листинге 3.1 представлена программная реализация описанных классов и функций.

3.3 Тестирование программы

В таблице 3.1 и 3.2 представлены описания тестов по методологии чёрного ящика, все тесты пройдены успешно.

Таблица 3.1 — Описание тестов по методологии чёрного ящика

	Описание теста	**Входные данные**	**Ожидаемый результат**	**Полученный результат**
1	проверка на обработку не валидных данных	dhsux	оповещение о некорректности данных и завершение работы программы	оповещение о некорректности данных и завершение работы программы
2	операции над положительными числами меньше 2^{32}	123456 321654	first number: $1^1 2^6 3^1 643^1$ second number: $1^1 2^1 3^1 53609^1$ multiplication result: $1^2 2^7 3^2 643^1 53609^1$ div: 0 mod: $1^1 2^5 643^1$ compare: < sum: $1^2 2^1 3^1 5^1 37^1 401^1$ diff: $-1^2 2^1 3^2 7^1 11^2 13^1$	first number: $1^1 2^6 3^1 643^1$ second number: $1^1 2^1 3^1 53609^1$ multiplication result: $1^2 2^7 3^2 643^1 53609^1$ div: 0 mod: $1^1 2^5 643^1$ compare: < sum: $1^2 2^1 3^1 5^1 37^1 401^1$ diff: $-1^2 2^1 3^2 7^1 11^2 13^1$
3	операции над положительными числами больше чем 2^{32}	5296967296 519696729	first number: $1^1 2^7 41382557^1$ second number: $1^1 3^4 2161^1 2969^1$ multiplication result: $1^2 2^7 3^4 2161^1 2969^1 41382557^1$ div: $1^1 2^1 5^1$ mod: $1^1 2^1 491^1 101833^1$ compare: > sum: $1^2 5^2 232666561^1$ diff: $1^2 19^1 139^1 1808887^1$	first number: $1^1 2^7 41382557^1$ second number: $1^1 3^4 2161^1 2969^1$ multiplication result: $1^2 2^7 3^4 2161^1 2969^1 41382557^1$ div: $1^1 2^1 5^1$ mod: $1^1 2^1 491^1 101833^1$ compare: > sum: $1^2 5^2 232666561^1$ diff: $1^2 19^1 139^1 1808887^1$
4	проверка на корректность операций над нулями	0 0	first number: 0 second number: 0 multiplication result: 0 div: DIVISION ERROR mod: DIVISION ERROR compare: == sum: 0 diff: 0	first number: 0 second number: 0 multiplication result: 0 div: DIVISION ERROR mod: DIVISION ERROR compare: == sum: 0 diff: 0

Таблица 3.2 — Описание тестов по методологии чёрного ящика

	Описание теста	**Входные данные**	**Ожидаемый результат**	**Полученный результат**
5	проверка корректности операций при одном отрицательном числе	-34534 -2345	second number: $-1^1 5^1 7^1 67^1$ multiplication result: $1^2 2^1 5^1 7^1 31^1 67^1 557^1$ div: $1^1 2^1 7^1$ mod: $1^1 2^3 3^1 71^1$ compare: < sum: $-1^2 3^1 19^1 647^1$ diff: $-1^2 32189^1$	second number: $-1^1 5^1 7^1 67^1$ multiplication result: $1^2 2^1 5^1 7^1 31^1 67^1 557^1$ div: $1^1 2^1 7^1$ mod: $1^1 2^3 3^1 71^1$ compare: < sum: $-1^2 3^1 19^1 647^1$ diff: $-1^2 32189^1$
6	проверка на корректность операций с одним отрицательным числом	23452 -354	first number: $1^1 2^2 11^1 13^1 41^1$ second number: $-1^1 2^1 3^1 59^1$ multiplication result: $-1^2 2^3 3^1 11^1 13^1 41^1 59^1$ div: $-1^1 2^1 3^1 11^1$ mod: $-1^1 2^2 11^1$ compare: > sum: $1^2 2^1 11549^1$ diff: $1^2 2^1 11903^1$	first number: $1^1 2^2 11^1 13^1 41^1$ second number: $-1^1 2^1 3^1 59^1$ multiplication result: $-1^2 2^3 3^1 11^1 13^1 41^1 59^1$ div: $-1^1 2^1 3^1 11^1$ mod: $-1^1 2^2 11^1$ compare: > sum: $1^2 2^1 11549^1$ diff: $1^2 2^1 11903^1$

ЗАКЛЮЧЕНИЕ

Была достигнута цель работы, освоены принципы длинной арифметики.

Для достижения поставленной цели были успешно выполнены основные задачи:

1) описаны принципы переведения чисел из десятичной системы счислений в систему состоящую из множителей числа,
2) описаны принципы операций в данной системе счислений,
3) реализованы операции над длинными числами,
4) реализованная программа успешно протестирована.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Иванов Д.В. О решете Эратосфена и гипотезе Коллатца. — Самара: Интернаука, 2020. — С. 12-15.

2. Длинная арифметика. [Электронный ресурс]. URL: https://wiki.algocode.ru/index.php?title=Длинная_арифметика (дата обращения: 26.12.2024).

Приложение А

Листинг 3.1 — Программная реализация описанных алгоритмов

```
#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <cmath>
#include <algorithm>
#include <cstdint>
using namespace std;

vector<unsigned long long> divs;
unsigned long limit = pow(2, 22);

void FIND_DIVS() {
  vector<bool> is_prime(limit + 1, true);
  is_prime[0] = is_prime[1] = false;

  for (unsigned long i = 2; i <= limit; ++i) {
    if (is_prime[i]) {
      divs.push_back(i);
      for (unsigned long j = 2 * i; j <= limit; j += i) {
        is_prime[j] = false;
      }
    }
  }
}

class Number_in_factors {
  public:
  map<int, int> number{ pair<int, int>{1, 1} };
  int negative = 1;
  bool empty = 0;

  void convert_to_factors(string inp) {
    unsigned long long num_10 = 0;
    int digit = inp.size() - 1;
    for (int i = 0; i < inp.size(); i++) {
      if ((inp[i] == '-') and (negative == 1)) {
        negative = -1;
        digit--;
```

```
40            continue;
41          }
42          else if (inp[i] == '-') {
43            cout << "ERRROR!CHECK THE OPERATORS.\n";
44            exit(1);
45          }
46          else if (inp[i] == '0') {
47            digit--;
48            continue;
49          }
50          string loc(1, inp[i]);
51          num_10 += stoi(loc) * pow(10, digit);
52          digit--;
53        }
54        if (num_10 == 0) {
55          empty = 1;
56          number.erase(1);
57          return;
58        }
59        empty = 0;
60        int loc_ind = 0;
61        while ((num_10 != 1) and (loc_ind < divs.size())) {
62          if (num_10 % divs[loc_ind] == 0) {
63            number[divs[loc_ind]] = 1;
64            num_10 /= divs[loc_ind];
65            while (num_10 % divs[loc_ind] == 0) {
66              number[divs[loc_ind]] += 1;
67              num_10 /= divs[loc_ind];
68            }
69          }
70          loc_ind++;
71        }
72        if (loc_ind >= divs.size()) {
73          cout << "IT IS TOO MUCH(PROGRAM STARTED EXTRA CALCULATES) ";
74          for (unsigned long long int i = divs[loc_ind - 1]; i <= num_10;
                i++) {
75            if (num_10 % i == 0) {
76              number[i] = 1;
77              num_10 /= i;
78              while (num_10 % i == 0) {
79                number[i] += 1;
```

```
            num_10 /= i;
          }
          if (num_10 == 1) break;
        }
      }
    }

    void print() {
      if (empty) cout << "0";
      else if (negative == -1) cout << "- ";
      for (auto it = number.begin(); it != number.end(); ++it) {
        cout << it->first << "^" << it->second << " ";
      }
      cout << "\n";
    }

    Number_in_factors operator*(const Number_in_factors _2) {
      Number_in_factors res;
      res.negative = this->negative * _2.negative;

      res.number = this->number;

      for (auto it = _2.number.begin(); it != _2.number.end(); ++it) {
        if (res.number.find(it->first) == res.number.end()) res.number[
           it->first] = it->second;
        else res.number[it->first] += it->second;
      }
      if ((empty or _2.empty)) {
        res.empty = 1;
        res.number.clear();
      }
      return res;
    }

    Number_in_factors operator+(const Number_in_factors _2) {
      Number_in_factors common_factors;
      long long small_sum_1 = this->empty == 1 ? 0 : 1 * this->negative
         ;
      long long small_sum_2 = _2.empty == 1 ? 0 : 1 * _2.negative;

```

```
119       for (auto it = _2.number.begin(); it != _2.number.end(); ++it) {
120         if (this->number.find(it->first) != this->number.end()) {
121           common_factors.number[it->first] = min(this->number[it->first
                ], it->second);
122           if (this->number[it->first] > it->second) small_sum_1 *= pow(
                it->first, (this->number[it->first] - it->second));
123           else if (this->number[it->first] < it->second) small_sum_2 *=
                 pow(it->first, (it->second - this->number[it->first]));
124         }
125         else small_sum_2 *= pow(it->first, it->second);
126       }
127       for (auto it = this->number.begin(); it != this->number.end(); ++
             it) {
128         if (_2.number.find(it->first) == _2.number.end()) {
129           small_sum_1 *= pow(it->first, it->second);
130         }
131       }
132
133       Number_in_factors small_sum;
134       if ((small_sum_1 + small_sum_2) == 0) {
135         small_sum.empty = 1;
136       }
137       if ((small_sum_1 + small_sum_2) < 0) small_sum.negative = -1;
138       small_sum.convert_to_factors(to_string(abs(small_sum_1 +
             small_sum_2)));
139       return common_factors*small_sum;
140     }
141
142     Number_in_factors operator-(const Number_in_factors _2) {
143       Number_in_factors common_factors;
144       long long small_sum_1 = this->empty == 1 ? 0 : 1*this->negative;
145       long long small_sum_2 = _2.empty == 1 ? 0 : 1*_2.negative;
146
147       for (auto it = _2.number.begin(); it != _2.number.end(); ++it) {
148         if (this->number.find(it->first) != this->number.end()) {
149           common_factors.number[it->first] = min(this->number[it->first
                ], it->second);
150           if (this->number[it->first] > it->second) small_sum_1 *= pow(
                it->first, (this->number[it->first] - it->second));
151           else if (this->number[it->first] < it->second) small_sum_2 *=
                 pow(it->first, (it->second - this->number[it->first]));
```

```
152         }
153         else small_sum_2 *= pow(it->first, it->second);
154       }
155       for (auto it = this->number.begin(); it != this->number.end(); ++
             it) {
156         if (_2.number.find(it->first) == _2.number.end()) {
157           small_sum_1 *= pow(it->first, it->second);
158         }
159       }
160
161       Number_in_factors small_sum;
162       if ((small_sum_1 - small_sum_2) == 0) {
163         small_sum.empty = 1;
164         small_sum.number.clear();
165         return small_sum;
166       }
167       if ((small_sum_1 - small_sum_2) < 0) small_sum.negative = -1;
168       if (small_sum_1 > small_sum_2) small_sum.convert_to_factors(
             to_string(abs(small_sum_1 - small_sum_2)));
169       else if (small_sum_1 < small_sum_2) small_sum.convert_to_factors(
             to_string(abs(small_sum_1 - small_sum_2)));
170       return common_factors * small_sum;
171     }
172   };
173
174   vector<unsigned long long> cancellation(Number_in_factors _1,
        Number_in_factors _2) {//num, num -> 10, 10 unsigned
175     vector<unsigned long long> res(2, 1);
176     int range = 0;
177     res[0] = _1.empty == 1 ? 0 : 1;
178     res[1] = _2.empty == 1 ? 0 : 1;
179
180     try {
181       for (auto it = _2.number.begin(); it != _2.number.end(); ++it) {
182         if (_1.number.find(it->first) == _1.number.end())  res[1] *=
               pow(it->first, it->second);
183         else {
184           range = _1.number[it->first] - it->second;
185           if (range > 0) res[0] *= pow(it->first, range);
186           else if (range < 0) res[1] *= pow(it->first, abs(range));
187           _1.number.erase(it->first);
```

```
188         }
189       }
190
191       for (auto it = _1.number.begin(); it != _1.number.end(); ++it) {
192         res[0] *= pow(it->first, it->second);
193       }
194     }
195     catch (...) {
196       cout << "ERROR IN DIV\n";
197     }
198
199     return res;
200   }
201
202   Number_in_factors div(Number_in_factors* _1, Number_in_factors* _2,
        string mode) {
203     int sign = _1->negative * _2->negative;
204     vector<unsigned long long> frac(2, 1);
205     frac = cancellation(*_1, *_2);
206
207     Number_in_factors res;
208     if (_2->empty) {
209       res.empty = 1;
210       res.number.erase(1);
211       cout << "DIVISION ERROR ";
212       return res;
213     }
214     string a;
215     if (mode == "div") {
216       res.negative = sign;
217       a = to_string(frac[0] / frac[1]);
218       res.convert_to_factors(a);
219     }
220     else {
221       res.negative = sign;
222       a = to_string(frac[0] % frac[1]);
223       res.convert_to_factors(a);
224     }
225     return res;
226   }
227
```

```
228   int compare(Number_in_factors _1, Number_in_factors _2) {// -1 -- <;
          1 -- >; 0 -- ==
229     if ((_1.negative < 0) and (_2.empty == 1)) return -1;// -1 0
230     if ((_1.negative < 0) and (_2.negative > 0)) return -1;// -1 1
231
232     if ((_1.empty == 1) and (_2.empty == 1)) return 0;// 0 0
233     if ((_1.empty == 1) and (_2.negative > 0)) return -1;// 0 1
234     if ((_1.empty == 1) and (_2.negative < 0)) return 1;// 0 -1
235
236     if ((_1.negative > 0) and (_2.negative < 0)) return 1;// 1 -1
237     if ((_1.negative > 0) and (_2.empty == 1)) return 1; // 1 0
238     // -1 -1
239     // 1 1
240
241     vector<Number_in_factors> res(2);
242     int range = 0;
243     try {
244       for (auto it = _2.number.begin(); it != _2.number.end(); ++it) {
245         if (_1.number.find(it->first) == _1.number.end()) res[1].number
              [it->first] = it->second;
246         else {
247           range = _1.number[it->first] - it->second;
248           if (range > 0) res[0].number[it->first] = range;
249           else if (range < 0) res[1].number[it->first] = abs(range);
250           _1.number.erase(it->first);
251         }
252       }
253
254       for (auto it = _1.number.begin(); it != _1.number.end(); ++it) {
255         res[0].number[it->first] = it->second;
256       }
257     }
258     catch (...) {
259       cout << "ERROR IN COMPARE\n";
260     }
261
262     unsigned long lg_1 = 0;
263     unsigned long lg_2 = 0;
264
265     for (auto it = res[0].number.begin(); it != res[0].number.end(); ++
          it) {
```

```
266         lg_1 += it->second * log(it->first);
267       }
268       for (auto it = res[1].number.begin(); it != res[1].number.end(); ++
              it) {
269         lg_2 += it->second * log(it->first);
270       }
271
272       if (_1.negative == -1) {
273         if (lg_1 > lg_2) return -1;
274         if (lg_1 < lg_2) return 1;
275       }
276       else {
277         if (lg_1 > lg_2) return 1;
278         if (lg_1 < lg_2) return -1;
279       }
280       return 0;
281     }
282
283     int main(){
284       Number_in_factors A, B, C;
285       string input_1, input_2;
286       FIND_DIVS();
287
288       cout << "enter the first number: ";
289       cin >> input_1;
290       A.convert_to_factors(input_1);
291       cout << "\nfirst number: ";
292       A.print();
293
294       cout << "enter the second number: ";
295       cin >> input_2;
296       B.convert_to_factors(input_2);
297       cout << "second number: ";
298       B.print();
299
300       C = A * B;
301       cout << "multiplication result: ";
302       C.print();
303
304       cout << "div: ";
305       div(&A, &B, "div").print();
```

```
306
307     cout << "mod: ";
308     div(&A, &B, "mod").print();
309
310     int r = compare(A, B);
311     cout << "compare: ";
312     if (r == -1) cout << "<\n";
313     if (r == 1) cout << ">\n";
314     if (r == 0) cout << "==\n";
315
316     cout << "sum: ";
317     C = A + B;
318     C.print();
319
320     cout << "diff: ";
321     C = A - B;
322     C.print();
323 }
```