Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

Отчёт по практическому занятию №2

«Арифметические операции с числами с плавающей точкой»

Выполнил:

студент гр. 153505

Савончик Е.В.

Проверила:

Калиновская А.А.

МИНСК 2023

**СОДЕРЖАНИЕ**

[**1.** **Цель работы** 3](#_Toc129246651)

[**2.** **Постановка задачи** 3](#_Toc129246652)

[**3.** **Теоретические сведения** 3](#_Toc129246653)

[**4.** **Результаты работы программы** 5](#_Toc129246654)

[**5.** **Выводы** 6](#_Toc129246655)

[**Приложение 1. Исходный код программы.** 7](#_Toc129246656)

# **Цель работы**

Рассмотреть представление чисел с плавающей точной в двоичном коде. Изучить алгоритмы выполнения основных арифметических операций над действительными числами с плавающей точкой. Написать программу, реализующую такие алгоритмы.

# **Постановка задачи**

***Задание к лабораторной работе 4***

Написать программу эмулятора АЛУ, реализующего *операции сложения и вычитания с плавающей*точкой  над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

***Задание к лабораторной работе 5***

Написать программу эмулятора АЛУ, реализующего *операцию умножения с плавающей точкой*над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

***Задание к лабораторной работе 6***

Написать программу эмулятора АЛУ, реализующего *операцию деления с плавающей точкой* над двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

# **Теоретические сведения**

В формате с фиксированной точкой, в частности в дополнительном коде, можно представлять положительные и отрицательные числа в диапазоне, симметричном на числовой оси относительно точки 0. Расположив воображаемую Разделяющую точку в середине разрядной сетки, можно в этом формате представлять не только целые, но и смешанные числа, а также дроби.

Однако такой подход позволяет представить на ограниченной разрядной сетке множество вещественных чисел в довольно узком диапазоне. Нельзя представить очень большие числа или очень маленькие. При выполнении деления двух больших чисел, как правило, теряется дробная часть частного.

При работе в десятичной системе счисления ученые давно нашли выход из положения, применяя для представления числовых величин так называемую научную нотацию. Так, число 976 000000 000 000 можно представить в виде 9.76x1014, а число 0,000000 000 000 0976 - в виде 9.76x10-14. При этом, фактически, разделительная точка динамически сдвигается в удобное место, а для того чтобы "уследить" за ее положением в качестве второго множителя - характеристики, - используется степень числа 10 (основания характеристики). Это позволяет с помощью небольшого числа цифр (т.е. чисел с ограниченной разрядностью) с успехом представлять как очень большие, так и очень малые величины.

# **Результаты работы программы**

В качестве средств для написания программы использовался язык программирования C++.

На рисунках представлен результат работы программы с разными входными данными.

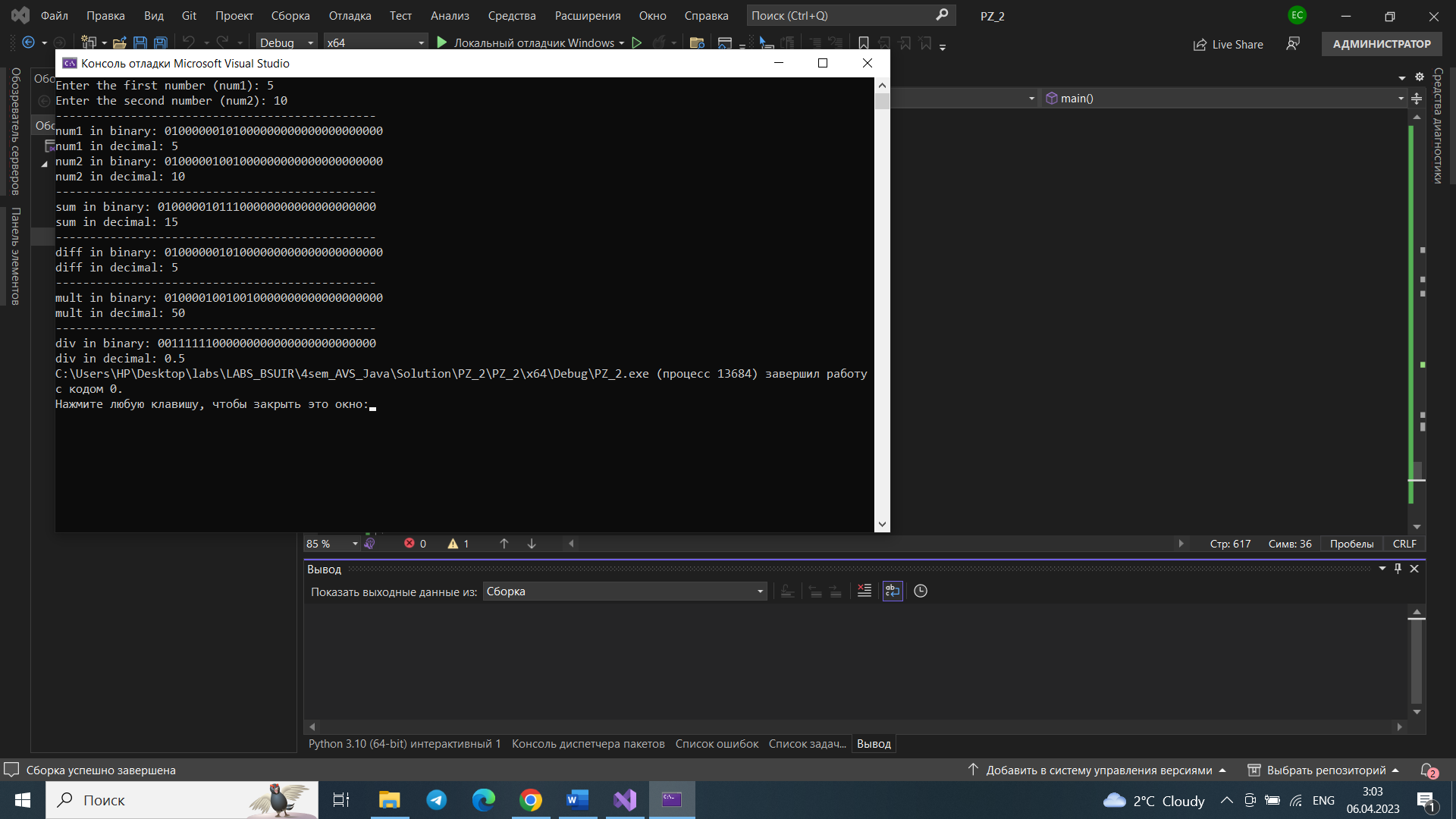


Рис. 1 – Пример (два положительных числа).

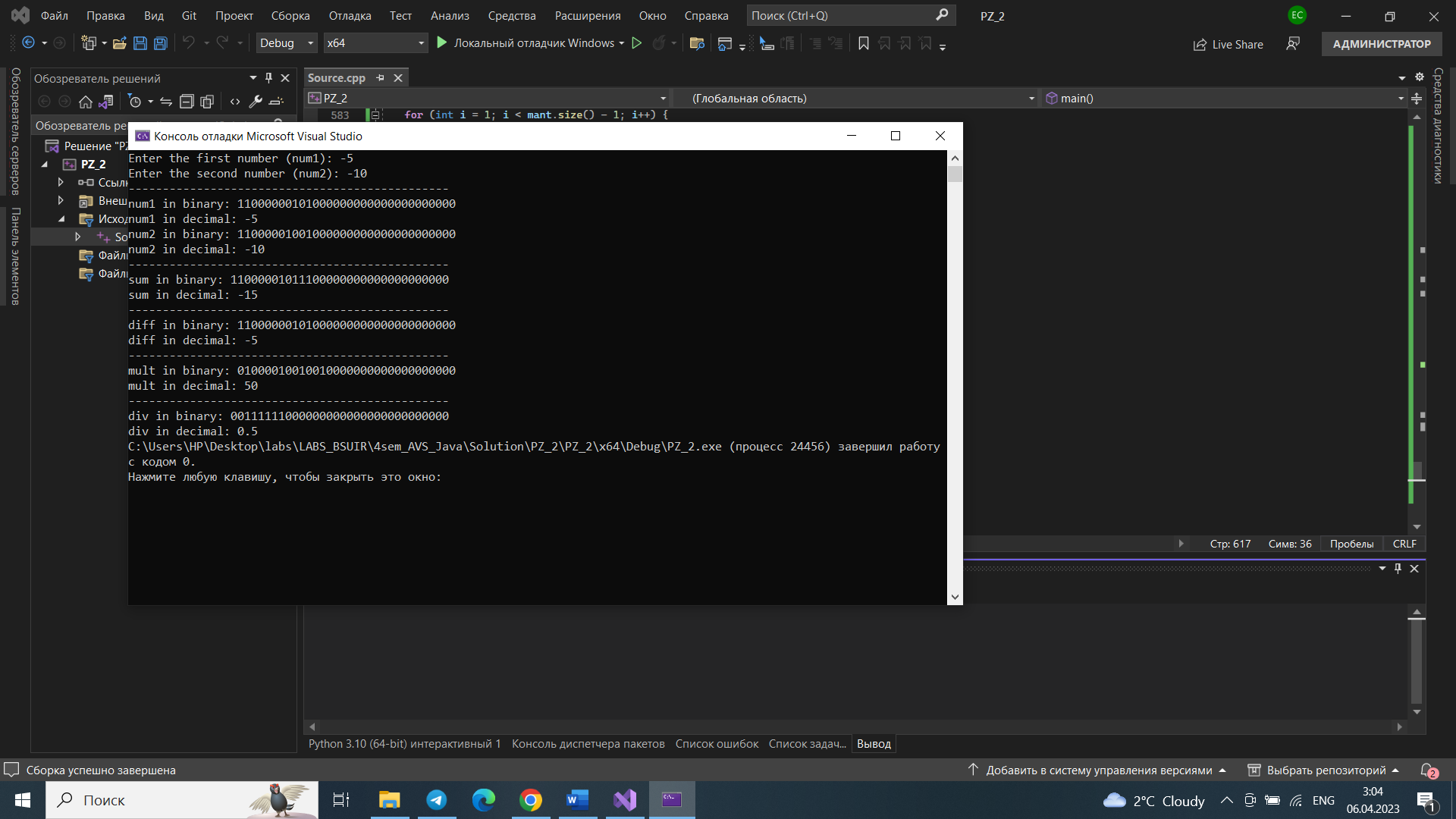


Рис. 2 – Пример (два отрицательных числа).

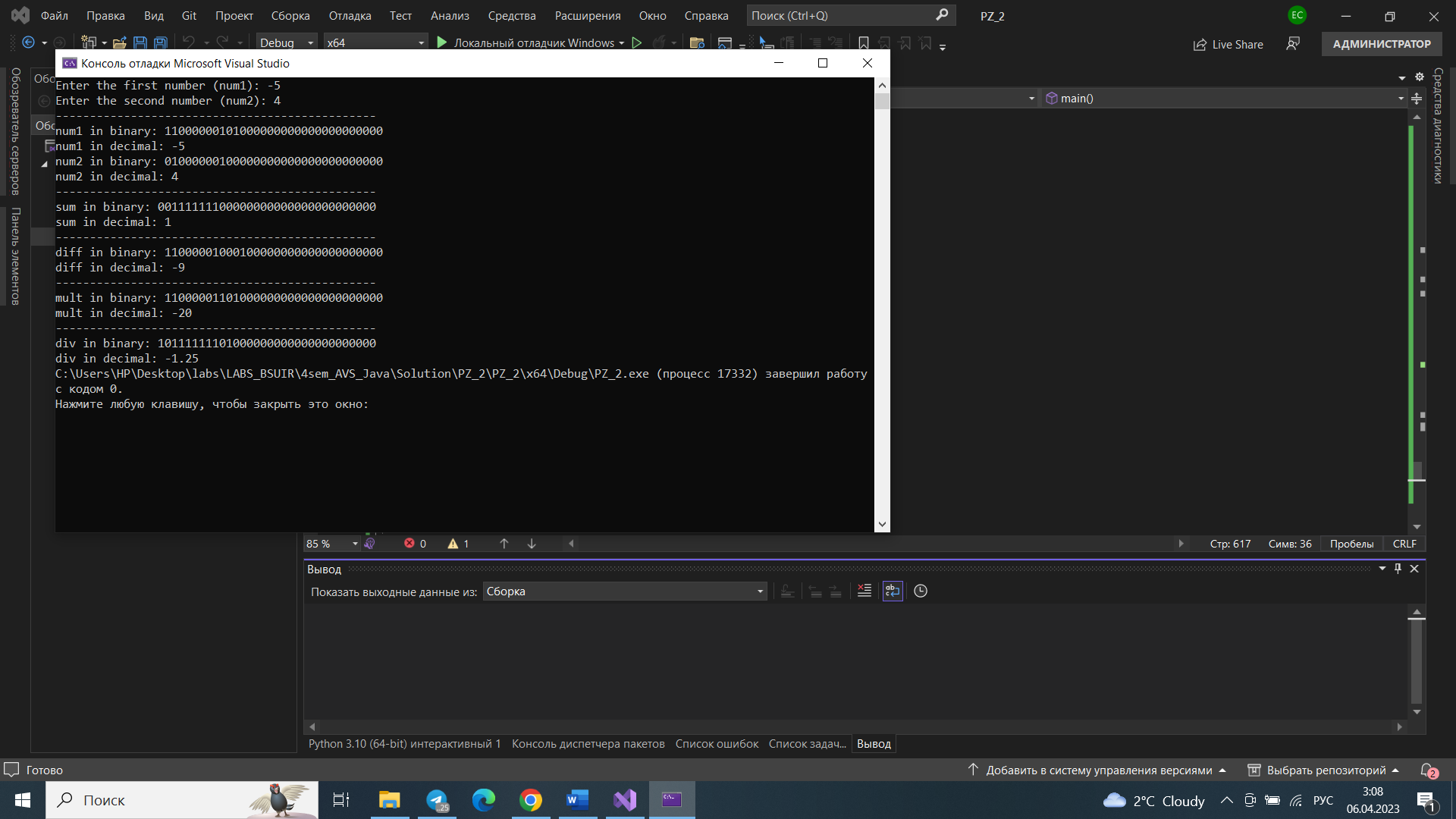


Рис. 3 – Пример (числа разного знака).

# **Выводы**

В ходе выполнения лабораторной было подробно изучено хранение дробных чисел в памяти компьютера. Так же была рассмотрена работа арифметико-логического устройства.

# **Приложение 1. Исходный код программы.**

#include <iostream>

#include <sstream>

#include <bitset>

#include <cmath>

#include <algorithm>

std::string binary\_difference(std::string num1\_s, std::string num2\_s, bool is\_print = false) {

std::string res;

int carry = 0, tmp;

std::reverse(num1\_s.begin(), num1\_s.end());

std::reverse(num2\_s.begin(), num2\_s.end());

for (size\_t i = 0; i < num1\_s.size(); i++) {

if (num1\_s[i] == '.') {

res += '.';

continue;

}

tmp = (num1\_s[i] - num2\_s[i] - carry);

if (tmp < 0) {

tmp = std::abs(tmp);

carry = 1;

res += (tmp%2) + 48;

} else {

res += tmp + 48;

carry = 0;

}

if (is\_print)

std::cout << res << std::endl;

}

std::reverse(res.begin(), res.end());

return res;

}

std::string binary\_summarize(std::string num1\_s, std::string num2\_s, bool is\_print = false) {

std::string res;

int carry = 0, tmp;

std::reverse(num1\_s.begin(), num1\_s.end());

std::reverse(num2\_s.begin(), num2\_s.end());

for (size\_t i = 0; i < num1\_s.size(); i++) {

if (num2\_s[i] == '.') {

res += '.';

continue;

}

tmp = (num1\_s[i] + num2\_s[i] + carry) - 96;

carry = tmp / 2;

res += std::to\_string(tmp % 2);

if (is\_print)

std::cout << res << std::endl;

}

std::reverse(num1\_s.begin(), num1\_s.end());

std::reverse(num2\_s.begin(), num2\_s.end());

if (carry) {

res += '1';

}

std::reverse(res.begin(), res.end());

return res;

}

std::string binary\_multiply(std::string num1\_s, std::string num2\_s, bool is\_print = false) {

std::string res, tmp\_res, null\_string(num1\_s.size() \* 2, '0');

res = null\_string;

if (num1\_s[0] == '0' && num2\_s[0] == '1')

std::swap(num1\_s, num2\_s);

std::reverse(num1\_s.begin(), num1\_s.end());

std::reverse(num2\_s.begin(), num2\_s.end());

for (size\_t i = 0; i < num2\_s.size(); i++) {

if (num2\_s[i] == '0')

continue;

tmp\_res = null\_string;

for (size\_t j = i; j < tmp\_res.size(); j++) {

tmp\_res[j] = '0';

}

for (size\_t j = i; j < i + num1\_s.size(); j++) {

tmp\_res[j] = num1\_s[j - i];

}

std::reverse(res.begin(), res.end());

std::reverse(tmp\_res.begin(), tmp\_res.end());

res = binary\_summarize(res, tmp\_res);

std::reverse(res.begin(), res.end());

if (is\_print)

std::cout << res << std::endl;

}

std::reverse(res.begin(), res.end());

return res;

}

std::string binary\_divide(std::string num1\_s, std::string M, bool is\_print = false) {

if (M.find("1") == std::string::npos)

return "divide by zero";

std::string res, tmp, A, Q, old\_a;

Q = num1\_s;

for (size\_t i = 0; i < 8; i++) {

M = M[0] + M;

}

for (size\_t i = 0; i < Q.size(); i++)

A += '0';

for (size\_t i = 0; i < Q.size(); i++) {

for (size\_t j = 0; j < A.size() - 1; j++)

A[j] = A[j + 1];

A[A.size() - 1] = Q[0];

for (size\_t j = 0; j < Q.size() - 1; j++)

Q[j] = Q[j + 1];

old\_a = A;

if (A[0] == M[0])

A = binary\_difference(A, M);

else

A = binary\_summarize(A, M);

if (A[0] == old\_a[0]) {

Q[Q.size() - 1] = '1';

} else {

Q[Q.size() - 1] = '0';

A = old\_a;

}

if (is\_print)

std::cout << std::to\_string(i) + ": " << A << ' ' << Q << std::endl;

}

return Q;

}

void inc(std::string &num\_s) {

int carry = 1, tmp;

std::reverse(num\_s.begin(), num\_s.end());

for (size\_t i = 0; i < num\_s.size(); i++) {

tmp = num\_s[i] - 48 + carry;

num\_s[i] = tmp % 2 + 48;

carry = tmp / 2;

}

if (carry == 1) {

num\_s += '1';

}

std:reverse(num\_s.begin(), num\_s.end());

}

void dec(std::string &num\_s) {

int carry = 1, tmp;

std::reverse(num\_s.begin(), num\_s.end());

for (auto &x:num\_s) {

tmp = x - 48 - carry;

if (tmp < 0) {

carry = 1;

x = tmp \* (-1) % 2 + 48;

} else {

x = tmp + 48;

carry = 0;

}

}

std::reverse(num\_s.begin(), num\_s.end());

}

std::string int\_part\_to\_binary(std::string num\_s) {

std::stringstream cont;

int num, size;

std::string res;

cont << num\_s;

cont >> num;

size = std::ceil(std::log2(num));

while (num) {

res += num % 2 + 48;

num /= 2;

}

std::reverse(res.begin(), res.end());

return res;

}

std::string complete\_fractional(std::string num\_s, size\_t zero\_count){

auto pos\_it = std::find(num\_s.begin(), num\_s.end(), '.');

size\_t k;

std::string zero\_str(zero\_count, '0');

num\_s += '.' + zero\_str;

return num\_s;

}

std::string fractional\_to\_binary(std::string fract\_part\_s, size\_t digit\_count) {

std::string res;

std::stringstream cont;

double fract\_part;

cont << fract\_part\_s;

cont >> fract\_part;

for (size\_t i = 0; i < digit\_count; i++) {

fract\_part \*= 2;

if (fract\_part >= 1) {

res += '1';

fract\_part -=1;

} else {

res += '0';

}

}

return res;

}

std::string decimal\_to\_binary(std::string num\_s) {

std::stringstream tmp\_cont;

double tmp\_num;

tmp\_cont << num\_s;

tmp\_cont >> tmp\_num;

if(tmp\_num == 0)

num\_s = "0";

if (num\_s[0] == '-')

num\_s.erase(0,1);

size\_t pos = std::distance(num\_s.begin(), std::find(num\_s.begin(), num\_s.end(), '.'));

if (pos == num\_s.size()) {

std::string res = int\_part\_to\_binary(num\_s);

res = complete\_fractional(res, 24 - res.size());

return res;

} else {

std::string int\_part\_s, fract\_part\_s, res;

double fract\_part;

int\_part\_s = num\_s.substr(0, pos);

fract\_part\_s = num\_s.substr(pos);

res = int\_part\_to\_binary(int\_part\_s);

std::stringstream cont;

res += '.';

res += fractional\_to\_binary(fract\_part\_s, 25-res.size());

return res;

}

}

std::string binary\_to\_float(std::string bin) {

int pos = std::distance(bin.begin(), std::find(bin.begin(), bin.end(), '.')), exp;

std::string exp\_s, mant, res;

if (pos == 1) {

if (bin[0] == '0') {

pos -= std::distance(bin.begin(), std::find(bin.begin(), bin.end(), '1'));

exp = 127 + pos;

exp\_s = std::bitset<8>(exp).to\_string();

pos = std::distance(bin.begin(), std::find(bin.begin(), bin.end(), '1'));

bin.erase(pos, 1);

mant = bin.substr(pos - 1);

} else {

exp = 127;

exp\_s = std::bitset<8>(exp).to\_string();

bin.erase(1, 1);

mant = bin;

}

} else if (pos == 0) {

exp = 127;

exp -= bin.find('1');

bin.erase(0, bin.find('1'));

exp\_s = std::bitset<8>(exp).to\_string();

mant = bin;

} else {

exp = 127 + pos - 1;

exp\_s = std::bitset<8>(exp).to\_string();

bin.erase(pos, 1);

mant = bin;

}

res = "0" + exp\_s + mant;

return res;

}

std::string decimal\_to\_float(std::string num\_s) {

bool is\_neg = 0;

std::string num\_b, res;

if (num\_s[0] == '-') {

is\_neg = 1;

num\_s.erase(0, 1);

}

num\_b = decimal\_to\_binary(num\_s);

res = binary\_to\_float(num\_b);

res[0] = is\_neg + 48;

return res;

}

std::string float\_summarize(std::string num1\_s, std::string num2\_s);

std::string float\_difference(std::string num1\_s, std::string num2\_s) {

std::string exp1 = num1\_s.substr(1, 8), exp2 = num2\_s.substr(1, 8),

mant1 = num1\_s.substr(9), mant2 = num2\_s.substr(9);

std::string mant;

char sign\_bit;

if (num1\_s[0] != num2\_s[0]) {

num2\_s[0] = (num2\_s[0] - 48) ^ 1 + 48;

return float\_summarize(num1\_s, num2\_s);

}

if (num1\_s.find('1') == std::string::npos) {

return num2\_s;

} else {

if (num2\_s.find('1') == std::string::npos)

return num1\_s;

}

if (num1\_s[0] == num2\_s[0] && (exp1 + mant1 > exp2 + mant2)) {

sign\_bit = num1\_s[0];

} else if (num1\_s[0] == num2\_s[0] && (exp1 + mant1 < exp2 + mant2)) {

sign\_bit = num2\_s[0];

}

if (exp1 + mant1 < exp2 + mant2) {

std::swap(exp1, exp2);

std::swap(mant1, mant2);

}

if (exp1 == exp2) {

mant = binary\_difference(mant1, mant2);

if (mant.find('1') == std::string::npos)

return std::string(33, '0');

while (!(mant[0]-48)) {

mant.erase(0, 1);

mant += '0';

dec(exp1);

}

if (mant.find('1') == std::string::npos)

return std::string(33, '0');

} else {

if (exp1 < exp2) {

std::swap(exp1, exp2);

std::swap(num1\_s, num2\_s);

std::swap(mant1, mant2);

}

while (exp1 != exp2 && mant2.find('1') != std::string::npos) {

inc(exp2);

mant2 = '0' + mant2;

mant2.erase(mant2.size() - 1, 1);

}

if (mant2.find('1') == std::string::npos) {

return num1\_s;

} else {

mant = binary\_difference(mant1, mant2);

if (mant.find('1') == std::string::npos)

return std::string(33, '0');

while (!(mant[0]-48)) {

mant.erase(0, 1);

mant += '0';

dec(exp1);

}

}

}

return sign\_bit + exp1 + mant;

}

std::string float\_summarize(std::string num1\_s, std::string num2\_s) {

if (num1\_s[0] != num2\_s[0]) {

if (num1\_s[0] == '1') {

std::swap(num1\_s, num2\_s);

num2\_s[0] = '0';

} else {

num2\_s[0] = '0';

}

return float\_difference(num1\_s, num2\_s);

}

std::string exp1 = num1\_s.substr(1, 8), exp2 = num2\_s.substr(1, 8),

mant1 = num1\_s.substr(9), mant2 = num2\_s.substr(9);

std::string mant;

if (num1\_s.find('1') == std::string::npos) {

return num2\_s;

} else {

if (num2\_s.find('1') == std::string::npos)

return num1\_s;

}

if (exp1 == exp2) {

mant = binary\_summarize(mant1, mant2);

if (mant.find('1') == std::string::npos) {

return std::string(33, '0');

}

if (mant.size() > 24) {

mant.erase(mant.size() - 1, 1);

inc(exp1);

if (exp1.size() > 8) {

std::cout << "perepolnenie poriadka";

exit(0);

}

}

} else {

if (exp1 < exp2) {

std::swap(exp1, exp2);

std::swap(num1\_s, num2\_s);

std::swap(mant1, mant2);

}

while (exp1 != exp2 && mant2.find('1') != std::string::npos) {

inc(exp2);

mant2 = '0' + mant2;

mant2.erase(mant2.size() - 1, 1);

}

if (mant2.find('1') == std::string::npos) {

return num1\_s;

} else {

mant = binary\_summarize(mant1, mant2);

if (mant.find('1') == std::string::npos)

return std::string(33, '0');

if (mant.size() > 24) {

mant.erase(mant.size() - 1, 1);

inc(exp1);

if (exp1.size() > 8) {

std::cout << "perepolnenie poriadka";

exit(0);

}

}

}

}

return num1\_s[0] + exp1 + mant;

}

std::string float\_multiply(std::string num1\_s, std::string num2\_s) {

std::string exp1 = '0' + num1\_s.substr(1, 8), exp2 = '0' + num2\_s.substr(1, 8),

mant1 = num1\_s.substr(9), mant2 = num2\_s.substr(9);

char sign\_bit = (num1\_s[0] - 48) ^ (num2\_s[0] - 48) + 48;

std::string mant, exp;

int pos1 = mant1.rfind('1'), pos2 = mant2.rfind('1');

mant1.erase(pos1 + 1);

mant2.erase(pos2 + 1);

mant1 = std::string(24 - mant1.size(), '0') + mant1;

mant2 = std::string(24 - mant2.size(), '0') + mant2;

exp = binary\_summarize(exp1, exp2);

if (exp.size() == 9) {

exp = binary\_difference(exp, std::bitset<9>(127).to\_string());

exp.erase(0, 1);

} else {

exp = binary\_difference(exp, std::bitset<8>(127).to\_string());

}

mant = binary\_multiply(mant1, mant2);

int pos = mant.find('1'), size1 = mant1.size() - mant1.find('1') - 1, size2 = mant2.size() - mant2.find('1') - 1,

size = mant.size() - mant.find('1') - 1;

if (size > size1 + size2) {

for (size\_t i = 0; i < size - size1 - size2; i++)

inc(exp);

} else if (size < size1 + size2) {

for (size\_t i = 0; i < size1 + size2 - size; i++)

dec(exp);

}

mant.erase(0, pos);

if (mant.size() <= 23) {

mant += std::string(24 - mant.size(), '0');

} else {

mant.erase(24);

}

while (mant[0] != '1') {

dec(exp);

mant.erase(0, 1);

mant += '0';

}

return sign\_bit + exp + mant;

}

std::string float\_divide(std::string num1\_s, std::string num2\_s) {

std::string exp1 = '0' + num1\_s.substr(1, 8), exp2 = '0' + num2\_s.substr(1, 8),

mant1 = num1\_s.substr(9), mant2 = num2\_s.substr(9);

std::string exp, mant;

char sign\_bit = (num1\_s[0] - 48) ^ (num2\_s[0] - 48) + 48;

int pres = 24;

exp = binary\_summarize(exp1, std::bitset<9>(127).to\_string());

exp = binary\_difference(exp, exp2);

mant1 += std::string(pres, '0');

mant2 = std::string(pres, '0') + mant2;

mant = binary\_divide(mant1, mant2);

exp.erase(0, 1);

int pos = mant.find('1');

if (pos < mant.size() - 1 - pres) {

for (size\_t i = 0; i < mant.size() - 1 - pres - pos; i++)

inc(exp);

} else {

for (size\_t i = 0; i < pos - (mant.size() - 1 - pres); i++)

dec(exp);

}

mant.erase(0, pos);

if (mant.size() > 24)

mant.erase(24);

return sign\_bit + exp + mant;

}

double float\_to\_decimal(std::string num\_s) {

std::string exp = num\_s.substr(1, 8), mant = num\_s.substr(9);

double res = 1;

for (int i = 1; i < mant.size() - 1; i++) {

res += (mant[i] - 48) \* std::pow(2.0, 0 - i);

}

res \*= std::pow(2, (int(std::bitset<8>(exp).to\_ulong()) - 127));

if (num\_s[0] - 48)

res \*= -1;

return res;

}

void print\_float(std::string float\_s) {

float\_s.erase(9, 1);

std::cout << float\_s;

}

int main()

{

std::string num1, num2, num1\_d, num2\_d, sum, diff, mul, div;

std::cout << "Enter the first number (num1): ";

std::cin >> num1;

std::cout << "Enter the second number (num2): ";

std::cin >> num2;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

print\_float(decimal\_to\_float(num1));

std::cout << " -- num1 in binary" << std::endl;

std::cout << float\_to\_decimal(decimal\_to\_float(num1)) << " -- num1 in decimal\n";

print\_float(decimal\_to\_float(num2));

std::cout << " -- num2 in binary" << std::endl;

std::cout << float\_to\_decimal(decimal\_to\_float(num2)) << " -- num2 in decimal\n";

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

sum = float\_summarize(decimal\_to\_float(num1), decimal\_to\_float(num2));

print\_float(sum);

std::cout << " -- summary in binary" << std::endl;

std::cout << float\_to\_decimal(sum) << " -- summary in decimal\n";

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

diff = float\_difference(decimal\_to\_float(num1), decimal\_to\_float(num2));

print\_float(diff);

std::cout << " -- difference in binary" << std::endl;

std::cout << float\_to\_decimal(diff) << " -- difference in decimal\n";

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

mul = float\_multiply(decimal\_to\_float(num1), decimal\_to\_float(num2));

print\_float(mul);

std::cout << " -- multiplication in binary" << std::endl;

std::cout << float\_to\_decimal(mul) << " -- multiplication in decimal\n";

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

div = float\_divide(decimal\_to\_float(num1), decimal\_to\_float(num2));

print\_float(div);

std::cout << " -- division in binary" << std::endl;

std::cout << float\_to\_decimal(div) << " -- division in decimal\n";

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

return 0;

}