Представление чисел с плавающей запятой. Стандарт IEEE 754

*Системы счисления*

Общепринятой является десятичная система счисления — позиционная система счисления, в основании которой лежит число 10. Числа записываются с помощью десяти цифр: 0, 1, 2 ,...,8 ,9. В десятичной записи числа каждая цифра имеет свой ≪вес≫, зависящий от позиции, в которой эта цифра стоит. Так, число 12502 можно представить в виде 1 • 104 + 2 • 103 + 5 • 102+ 0 • 101+2 • 100, видно, что цифры, стоящие в разных разрядах, имеют разный ≪вес≫.

Число можно представлять не только в десятичной, но и в иных системах счисления, например, двоичной или восьмеричной, разложив это число по степеням числа два или, соответственно, числа восемь. Рассмотрим пример: представить

число 183(10) в двоичной, восьмеричной и троичной системах счисления (нижний индекс 10 указывает на то, в какой системе счисления записано число).

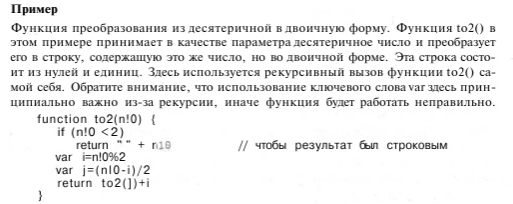
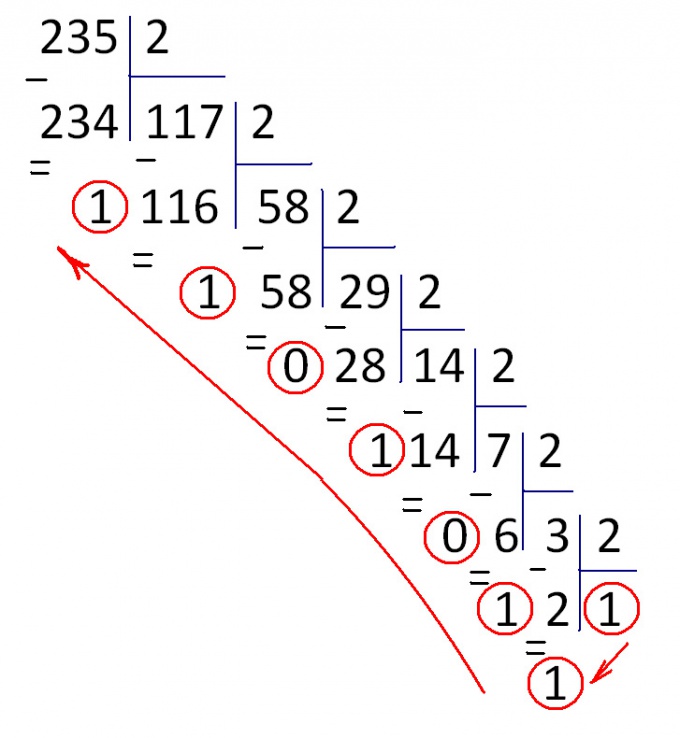
• 183= 128 +32 +16 +4 +2 +1 =27 +25 +24 +22 +21 +2°=10110111(2),

• 183 = 2 • 64 +6 • 8 +7 =2 • 82 + 6 • 81 + 7 • 8° = 267(8),

• 183 = 2 • 81 +2 • 9 +3 =2 • З4 +2 • З2 +1 • З1 = 20210(3).

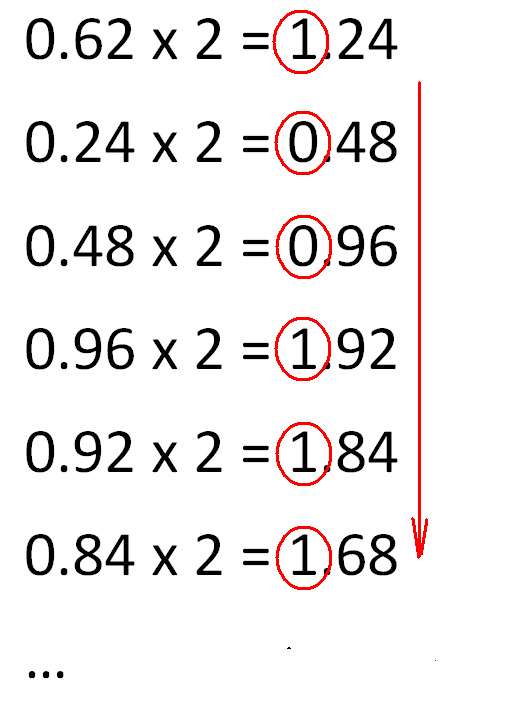
Инструкция.

Для того, чтобы перевести дробное десятичное число в двоичную систему счисления, действуйте по следующему алгоритму. Рассмотрим действие алгоритма на примере числа 235.62. Сначала переводится целая часть числа. Делим десятичное число на два до тех пор, пока не получим неделимый на два остаток. На каждом шаге деления получим остаток 1 (если делимое число было нечетным) или 0 (если делимое делится на два без остатка). Все эти остатки обязательно должны быть учтены. Последнее частное, полученное в результате такого пошагового деления, всегда будет единицей.  
Записываем последнюю единицу в старший разряд искомого двоичного числа, а полученные в процессе остатки записываем за этой единицей в обратном порядке. Здесь надо быть внимательным и не пропускать нули.  
Таким образом, числу 235 в двоичном коде будет соответствовать число 11101011.



3

Теперь переведем в двоичную систему счисления дробную часть десятичного числа. Для этого последовательно умножаем дробную часть числа на 2 и фиксируем целые части полученных чисел. Эти целые части дописываем к полученному в предыдущем шаге числу после двоичной точки в прямом порядке.  
Тогда десятичному дробному числу 235.62 соответствует двоичное дробное 11101011.100111.



Двоичная дробная часть числа будет конечной, только если дробная часть исходного числа конечна и заканчивается на 5. Простейший случай: 0.5 х 2 = 1, следовательно 0.5 в десятичной системе - это 0.1 в двоичной.

Для перевода двоичного числа в десятичное необходимо это число представить в виде суммы произведений степеней основания двоичной системы счисления на соответствующие цифры в разрядах двоичного числа.

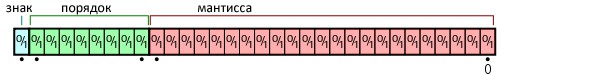
Например, требуется перевести двоичное число 10110110 в десятичное. В этом числе 8 цифр и 8 разрядов (разряды считаются, начиная с нулевого, которому соответствует младший бит). В соответствии с уже известным нам правилом представим его в виде суммы степеней с основанием 2:

101101102 = (1·27)+(0·26)+(1·25)+(1·24)+(0·23)+(1·22)+(1·21)+(0·20)=128+32+16+4+2 = 18210

**Представление чисел с плавающей запятой.**

**Нормализованные числа**

Множество целых чисел бесконечно, но мы всегда можем подобрать такое число бит, чтобы представить любое целое число, возникающее при решении конкретной задачи. Множество действительных чисел не только бесконечно, но еще и непрерывно, поэтому, сколько бы мы не взяли бит, мы неизбежно столкнемся с числами, которые не имеют точного представления. Числа с плавающей запятой — один из возможных способов представления действительных чисел, который является компромиссом между точностью и диапазоном принимаемых значений.  
  
Число с плавающей запятой состоит из набора отдельных разрядов, условно разделенных на знак, порядок и мантиссу. Порядок и мантисса — целые числа, которые вместе со знаком дают представление числа с плавающей запятой в следующем виде:  
  
  
  
Математически это записывается так:  
  
(-1)s × M × BE, где s — знак, B-основание, E — порядок, а M — мантисса.



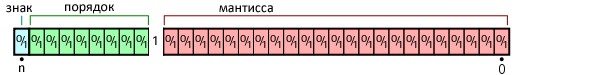
Основание определяет систему счисления разрядов. Математически доказано, что числа с плавающей запятой с базой B=2 (двоичное представление) наиболее устойчивы к ошибкам округления, поэтому на практике встречаются только базы 2 и, реже, 10. Для дальнейшего изложения будем всегда полагать B=2, и формула числа с плавающей запятой будет иметь вид:  
  
(-1)s × M × 2E

Что такое мантисса и порядок? *Мантисса* – это целое число фиксированной длины, которое представляет старшие разряды действительного числа. Допустим наша мантисса состоит из трех бит (|M|=3). Возьмем, например, число «5», которое в двоичной системе будет равно 1012. Старший бит соответствует 22=4, средний (который у нас равен нулю) 21=2, а младший 20=1. *Порядок* – это степень базы (двойки) старшего разряда. В нашем случае E=2. Такие числа удобно записывать в так называемом стандартном виде, например «1.01e+2». Сразу видно, что мантисса состоит из трех знаков, а порядок равен двум.   
  
Допустим мы хотим получить дробное число, используя те же 3 бита мантиссы. Мы можем это сделать, если возьмем, скажем, E=1. Тогда наше число будет равно   
  
1,01e+1 = 1×21+0×20+1×2-1=2+0,5=2,5

Здесь, поскольку E=1, степень двойки первого разряда (который идет перед запятой), равна «1». Два других разряда, расположенных правее (после запятой), обеспечивают вклад 2E-1 и 2E-2 (20 и 2-1 соответственно). Очевидно, что регулируя E одно и то же число можно представить по-разному. Рассмотрим пример с длиной мантиссы |M|=4. Число «2» можно представить в следующем виде:   
  
2 = 10 (в двоичной системе) = 1.000e+1 = 0.100e+2 = 0.010e+3. (E=1, E=2, E=3 соответственно)

Обратите внимание, что одно и то же число имеет несколько представлений. Это неудобно для оборудования, т.к. нужно учитывать множественность представления при сравнении чисел и при выполнении над ними арифметических операций. Кроме того, это не экономично, поскольку число представлений — конечное, а повторения уменьшают множество чисел, которые вообще могут быть представлены. Поэтому уже в самых первых машинах начали использовать трюк, делая первый бит мантиссы всегда положительным. Такое представление назвали *нормализованным*.

Это экономит один бит, так как неявную единицу не нужно хранить в памяти, и обеспечивает уникальность представления числа. В нашем примере «2» имеет единственное нормализованное представление («1.000e+1»), а мантисса хранится в памяти как «000», т.к. старшая единица подразумевается неявно. Но в нормализованном представлении чисел возникает новая проблема — в такой форме невозможно представить ноль.   
  
Строго говоря, нормализованное число имеет следующий вид:  
  
(-1)s × 1.M × 2E.  
  
Качество решения задач во многом зависит от выбора представления чисел с плавающей запятой.



В числах одинарной точности (float/single) порядок состоит из 8 бит, а мантисса – из 23. Эффективный порядок определяется как E-127. Например, число 0,15625 будет записано в памяти как



В этом примере:

* Знак s=0 (положительное число)
* Порядок E=011111002-12710 = -3
* Мантисса M = 1.012 (первая единица не явная)
* В результате наше число F = 1.012e-3 = 2-3+2-5 = 0,125 + 0,03125 = 0,15625

Под порядок отведено 8 бит, согласно стандарту IEEE 754, выбран диапазон от -127 до +128 со смещением 127; при этом нулевой байт и байт, состоящий из единиц, отведены для специальных целей (хранения денормализованных чисел и бесконечностей соответственно). Таким образом, 8 бит порядка позволяют хранить порядки от -126 (00000001) до +127 (11111110) с использованием сдвига 127.

Примеры. Для удобства восприятия бит знака, порядок и мантиссу будем отделять пробелами.

1 . Число 13=1101(2)=1.101(2) \* 23. Бит знака равен 0 . К порядку добавляем 127: 3 +127 =130=10000010(2). В мантиссе убираем ведущую единицу, получаем

Числу 13 соответствует 0 10000010 10100000000000000000000.

2. Число -6.75=-110.11(2)=-1.1011(2) • 22. Бит знака равен 1. К порядку добавляем 127: 2+127=129=10000001(2) В мантиссе убираем ведущую единицу, получаем

Числу -6.75 соответствует 1 10000001 10110000000000000000000.

3. Число 0.15625=0.00101(2)=1.01(2) • 2-3. Бит знака равен 0. К порядку добавляем 127: -3+127=124=01111100(2). В мантиссе убираем ведущую единицу, получаем .

Числу 0.15625 соответствует 0 01111100 010000000000000000000000.

4. Число 1=1(2) =1.0(2) \* 2°. Бит знака равен 0. К порядку добавляем 127: 0 +127=127=01111111(2). В мантиссе убираем ведущую единицу, получаем .

Числу 1 соответствует 0 01111111 00000000000000000000000.

5 Число 0.2=0.001100110011...(2) =1.100110011...(2) • 2-3. Бит знака равен 0. К порядку добавляем 127:-3+127 =124=01111100 (2). В мантиссе убираем ведущую единицу, получаем 10011001100110011001100 .

Числу 0.2 соответствует 0 01111100 10011001100110011001100.

6. Определить число с плавающей запятой, лежащее в четырёх соседних байтах:

1 10000010 10010000000000000000000

- Знаковый бит, равный 1 показывает, что число отрицательное.

- Порядок 10000010 в десятичном виде соответствует числу 130. Вычтя число 127 из 130, получим число 3.

- Теперь запишем мантиссу: 1,10010000000000000000000

- И, наконец, определим десятичное число: -1100,1(2)= -12,5(10)

7. Определить число с плавающей запятой, лежащее в четырёх соседних байтах:

1 10000110 01101000000000000000000

- Знаковый бит, равный 1 показывает, что число отрицательное.

- Экспонента 10000110 в десятичном виде соответствует числу 134. Вычтя число 127 из 134, получим число 7.

- Теперь запишем мантиссу: 1,01101000000000000000000

- И, наконец, определим десятичное число: -10110100(2)=-180(10)

**Особые значения чисел с плавающей точкой**

**Ноль (со знаком)**

В нормализованной форме числа с плавающей точкой невозможно представить ноль. Поэтому для его представления зарезервированы специальные значения мантиссы и порядка — число считается нулём, если все его биты, кроме знакового, равны нулю. При этом в зависимости от значения бита знака ноль может быть как положительным, так и отрицательным.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Знак** | | | | |
|  | **Порядок** | | | | | **Мантисса** | | | | | | | | | | |  |
| **0/1** | **0** | **0** | **0** | **0** | **0** | **1,** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **=** |

Формат float предусматривает возможность хранения +0 и -0, внутреннее представление 0 00000000 000000000000000000000000 и 1 00000000 000000000000000000000000 соответственно. Отметим, что если для целых наличие +0 и -0 — существенный недостаток формата, то для вещественных вовсе нет. Причина в том, что особо малые положительные числа, которые уже не удается сохранить, сбрасываются в +0 , а отрицательные — в -0. Эти две величины математик-программист может и должен интерпретировать по-разному. Здесь уместно вспомнить математический анализ, который учит, что стремление к +0 и к -0 - это не одно и то же!

**Неопределенность (*NaN*)**

**NaN** — это аббревиатура от фразы "*not a number*". NaN является результатом арифметических операций, если во время их выполнения произошла ошибка (примеры см. ниже). В IEEE 754 NaN представлен как число, в котором все двоичные разряды порядка — единицы, а мантисса не нулевая.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Знак** | | | | |
|  | **Порядок** | | | | | **Мантисса** | | | | | | | | | | |  |
| **0/1** | **1** | **1** | **1** | **1** | **1** | **1,** | **0/1** | **0/1** | **0/1** | **0/1** | **0/1** | **0/1** | **0/1** | **0/1** | **0/1** | **0/1** | **=** |

Любая операция с NaN возвращает NaN. При желании в мантиссу можно записывать информацию, которую программа сможет интерпретировать. Стандартом это не оговорено и мантисса чаще всего игнорируется.

**Бесконечности**

В число с плавающей запятой можно записать значение или . Как и нули со знаком, бесконечности позволяют получить хотя бы близкий к правильному результат вычисления в случае переполнения. Согласно стандарту IEEE 754 число с плавающей запятой считается равным бесконечности, если все двоичные разряды его порядка — единицы, а мантисса равна нулю. Знак бесконечности определяется знаковым битом числа.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Знак** | | | | |
|  | **Порядок** | | | | | **Мантисса** | | | | | | | | | | |  |
| **0/1** | **1** | **1** | **1** | **1** | **1** | **1,** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **=** |

**Денормализованные числа**

**Денормализованные числа** (англ. *denormalized/subnormal numbers*) - это способ увеличить количество представимых числом с плавающей запятой значений около нуля, дабы повысить точность вычислений. Каждое значение денормализованного числа меньше самого маленького **нормализованного** ("обычного") значения числа с плавающей запятой. Согласно стандарту, если порядок равен своему минимальному значению (все его биты — нули, а истинное значение порядка равно его сдвигу) и все биты мантиссы равны нулю, то это . Если же мантисса не равна нулю, то это число с порядком, на единицу большим минимального (все биты порядка, кроме младшего — нули) и данной мантиссой, **целая часть которой считается равной нулю, а не единице**.



То есть число с плавающей запятой, при учете вышесказанного, можно задать следующим образом:

* , если (*нормализованное число*)



* , если (*денормализованное число*)



Где — бит знака, — последовательность битов мантиссы, — значение порядка (с учетом сдвига), — минимальное значение порядка, используемое для записи чисел (1 — *сдвиг*) , — минимальное значение порядка, которое он в принципе может принять (все биты нули, 0 — *сдвиг*).



Хоть денормализованные числа и позволяют бороться с погрешностями и обрабатывать очень маленькие значения, за эти возможности приходится дорого платить. Ввиду сложности денормализованные числа крайне редко реализуют на аппаратном уровне - вместо этого используются программные реализации, работающие значительно медленнее.

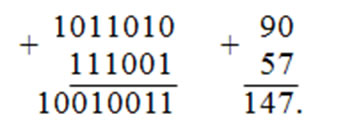
Арифметические действия в двоичной системе производится по тем же правилам что и в десятичной системе счисления. Однако так как в двоичной системе счисления используются только две цифры 0 и 1, то арифметические действия выполняются проще, чем десятичной системе.

**Сложение двоичных чисел.**

Сложение выполняется поразрядно столбиком, начиная с младшего разряда и используя таблицы двоичного сложения:

0 + 0 = 0  
0 + 1 = 1  
1 + 0 = 1  
1 + 1 = 10.

При сложении необходимо помнить, что 1+1 дают нуль в данном разряде и единицу переноса в старший.   
**Пример 3.5.** Сложить два числа:



Пример алгоритма сложения целых положительных двоичных чисел.

N1, N2 – двоичные числа в виде строки.

// Приведем к одинаковой разрядности.

Пока длинаN1>длина N2

N2=’0’+ N2

Пока длинаN2>длина N1

N1=’0’+ N1

Vstr=’’ // для сохранения результата сложения

n3=0 // для десяток

i= длина N1-1

Пока i>=0

Начало цикла

n4=Число(N1[i])+ Число(N2[i])+n3 // Складываем i разряд и десятки

Если (n4>1) // если n4=2, 3

n3=1

иначе

n3=0

Vstr =n4%2+ Vstr

i=i-1

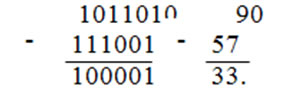
Конец цикла

Если (n3=1)

Vstr ='1'+ Vstr

**Вычитание двоичных чисел.**

Вычитание выполняется поразрядно столбиком, начиная с младшего разряда и используя таблицы двоичного  вычитания:  
0 – 0 = 0  
1 – 0 = 1  
1 – 1 = 0  
10 – 1 = 1.  
**Пример 3.6.** Найти разность двух чисел:



Т.е. при вычитании двоичных чисел в случае необходимости занимается 1 из старшего разряда, которая равна двум единицам младшего разряда.

**Сложение чисел с плавающей запятой**

Рассмотрим алгоритм сложения чисел с плавающей запятой для нормализованных чисел одного порядка, потом для чисел разного порядка.

Рассмотрим пример: требуется найти сумму 17 + 22.

Имеем 17(10) +22(10) = 1.0001∙24 +1.0110∙24 = 10.0111∙24=1.00111∙25. При сложении чисел одного порядка (в силу того, что ведущая цифра равна 1) сумма будет иметь порядок на 1 больший, чем у слагаемых. Таким образом, надо 1) вспомнить, что у мантисс ведущая цифра равна 1; 2) сложить мантиссы; 3) запятую в сумме сдвинуть на одну позицию влево (при этом 23-я цифра мантиссы выйдет за разрядную сетку) и порядок увеличить на 1; 4) ведущую единицу суммы мысленно выбросить.

Ведущие единицы мантиссы выделены курсивом.

17=0 10000011 *1* 00010000000000000000000

22=0 10000011 *1* 01100000000000000000000.

Складываем мантиссы:

39=0 10000011 *10* 01110000000000000000000.

Сдвигаем запятую в сумме на одну позицию влево и увеличиваем

порядок на 1:

39 =0 10000100 *1* 00111000000000000000000.

Мысленно выбрасываем ведущую единицу суммы:

39=0 10000100 00111000000000000000000 .

Правило сложения чисел с разными порядками гласит: *порядок меньшего слагаемого должен быть выравнен до порядка большего слагаемого за счет приписывания ведущих нулей к меньшему.*

Рассмотрим пример: требуется найти сумму 17+3.

Имеем 17(10)+3(10) = 1.0001∙24+1.1∙21 =1.0001∙24 + 0.0011∙24 =1.01 • 24.

Ведущие единицы мантиссы выделены курсивом.

17= 0 10000011 *1* 00010000000000000000000

3 = 0 10000000 *1* 10000000000000000000000.

Выравниваем порядки и пишем ведущие нули у меньшего слагаемого:

17= 0 10000011 *1* 00010000000000000000000

3 = 0 10000011 *0* 00110000000000000000000.

Складываем мантиссы:

20 = 0 10000011 *1* 01000000000000000000000.

Мысленно выбрасываем ведущую единицу суммы:

20 = 0 10000011 01000000000000000000000.

[**Как вычитать двоичные числа**](https://ru.wikihow.com/%D0%B2%D1%8B%D1%87%D0%B8%D1%82%D0%B0%D1%82%D1%8C-%D0%B4%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B5-%D1%87%D0%B8%D1%81%D0%BB%D0%B0)**.**

Этот метод называется «дополнение к двойке», так как замена цифр приводит к «дополнению к единице», а затем к полученному числу прибавляется 1.

Рассмотрим на примере.

17(10)=10001

-

3(10)=11

Приведем к одинаковой значимости (макс.значимость+1) приписав слева 0.

010001

-

000011

В вычитаемом числе поменяем цифры**:** каждую 1 поменяем на 0, а каждый 0 на 1. В нашем примере вычитаемое превращается в: ~~000011~~ **→** 111100.

К полученному вычитаемому прибавим 1**.** В нашем примере вы получите **111100 + 1 = 111101**.

Теперь вместо вычитания сложите два двоичных числа.

010001

+

111101

1001110

В полученном результате игнорируйте любую цифру, стоящую первой слева.

* + ~~1~~**001110 = 001110**
  + Таким образом, 10001-11=1110.
  + 1110=14(10).

Метод дополнения работает следующим образом: aaaaaa - bbbbbb = aaaaaa - (111111-cccccc)=aaaaaa-((1000000-1)-cccccc)=aaaaaa+(1+cccccc)-1000000.

**Постановка задачи**

Написать программу выполнения арифметических операций. Программа считывает из файла input.txt строку, содержащую два числа, разделенных операцией + или -. Программа переводит операнды во внутреннее представление и выполняет сложение/вычитание над операндами в их внутреннем представлении согласно правилам стандарта IEEE 754. В файл out.txt записывается результат выполнения арифметической операции во внутреннем и в десятичном виде.

Примеp.

|  |  |
| --- | --- |
| input.txt | 3.1+16.07 |
| out.txt | 3.1=0 10000000 10001100110011001100110  16.07=0 10000011 00000001000111101011100  0 10000011 00110010101110000101000  ≈19.169998168945312 |

|  |  |
| --- | --- |
| input.txt | 16.07 - 3.1 |
| out.txt | 16.07=0 10000011 00000001000111101011100  3.1=0 10000000 10001100110011001100110  0 10000010 10011111000010100100000  ≈12.970001220703125 |

|  |  |
| --- | --- |
| input.txt | 20.5 - 3.1 |
| out.txt | 20.5 = 0 10000011 01001000000000000000000  3.1 = 0 10000000 10001100110011001100110  0 10000011 00010110011001100110100  =17.400001525878906 |

|  |  |
| --- | --- |
| input.txt | 20.5 + 5.25 |
| out.txt | 20.5 = 0 10000011 01001000000000000000000  5.25 = 0 10000001 01010000000000000000000  0 10000011 10011100000000000000000  =25.75 |

|  |  |
| --- | --- |
| input.txt | 20.5 - 5.25 |
| out.txt | 20.5 = 0 10000011 01001000000000000000000  5.25 = 0 10000001 01010000000000000000000  0 10000010 11101000000000000000000  =15.25 |