Представление чисел в памяти

Борис Золотов (Б09, Б11)
Михаил Иванов (Б07, Б08)
Иван Казменко (Б01, Б05)
Владислав Макаров (Б02, Б03)
Мария Радионова (Б06)
Арина Филимонова (Б10)

Санкт-Петербургский Государственный Университет
Факультет математики и компьютерных наук

Среда, 1 октября 2025 года

Содержание

Содержание

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность.
- Краткость.
- Удобство арифметических действий.

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность.
- Краткость.
- Удобство арифметических действий.

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность: для маленьких чисел.
- Краткость: для очень маленьких чисел.
- Удобство арифметических действий: для маленьких чисел.

Пример 1 — унарная система счисления.
Число записывается палочками, стоящими подряд: |, ||, |||, ||||, ...

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность: до нескольких тысяч, нужна привычка.
- Краткость: до нескольких тысяч.
- Удобство арифметических действий: сложные правила.

Пример 2 — римские числа.
Вводятся специальные символы для обозначения больших количеств: $5 = V$, $10 = X$, $50 = L$, $100 = C$, $500 = D$, $1000 = M$.
Позиции символов также имеют значение: например, $VI = V + I = 6$, но $IV = V - I = 4$.

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность: для не очень больших чисел, нужна привычка.
- Краткость: для не очень больших чисел.
- Удобство арифметических действий: легко складывать и вычитать, с таблицей умножения можно производить умножение и деление в столбик.

Пример 3 — позиционные системы счисления: десятичная. Значение имеют и символ (цифра), и позиция (степень основания): $153_{10} = 3 \cdot 1 + 5 \cdot 10 + 1 \cdot 100$.

Системы счисления

Постановка задачи:
требуется записывать числа и производить операции с ними.

Критерии качества:

- Наглядность: для не очень больших чисел, нужна привычка.
- Краткость: для не очень больших чисел.
- Удобство арифметических действий: легко складывать и вычитать, таблица умножения простая, а значит, действия легко реализуются на уровне микросхемы.

Пример 4 — позиционные системы счисления: двоичная. Значение имеют и символ (цифра), и позиция (степень основания): $100101_2 = 1 + 4 + 32 = 37$.

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы компилятора GNU C++ для архитектуры x86:

Тип	**Байты**	**Биты**	**Минимум**	**Максимум**
`unsigned char?`	1	8	0	255
`unsigned short?`	2	16	0	65 535
`unsigned long?`	4	32	0	$2^{32} - 1$
`unsigned long long?`	8	64	0	$2^{64} - 1$

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы языка C++ (`#include <cstdint>`):

Тип	**Байты**	**Биты**	**Минимум**	**Максимум**
`std::uint8_t`	1	8	0	255
`std::uint16_t`	2	16	0	65 535
`std::uint32_t`	4	32	0	$2^{32} - 1$
`std::uint64_t`	8	64	0	$2^{64} - 1$

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы языка C++ (`#include <cstdint>`):

Тип	Байты	Биты	Минимум	Максимум
`std::uint8_t`	1	8	0	255
`std::uint16_t`	2	16	0	65 535
`std::uint32_t`	4	32	0	$2^{32} - 1$
`std::uint64_t`	8	64	0	$2^{64} - 1$

Адресовать память обычно можно с точностью до байта.
В каком порядке идут байты в целом числе?

- Big-endian: $1000 = 11\,1110\,1000_2$, код [0000 0011] [1110 1000]. Удобно для человеческого восприятия.
- Little-endian: $1000 = 11\,1110\,1000_2$, код [1110 1000] [0000 0011]. Если число помещается в меньший тип, адрес тот же.

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы компилятора GNU C++ для архитектуры x86:

Тип	**Байты**	**Биты**	**Минимум**	**Максимум**
`unsigned char?`	1	8	0	255
`unsigned short?`	2	16	0	65 535
`unsigned long?`	4	32	0	$2^{32} - 1$
`unsigned long long?`	8	64	0	$2^{64} - 1$
`signed char?`	1	8	-128	127
`signed short?`	2	16	$-32\,768$	32 767
`signed long?`	4	32	-2^{31}	$2^{31} - 1$
`signed long long?`	8	64	-2^{63}	$2^{63} - 1$

Для отрицательных чисел уменьшим диапазон на один бит. Этот бит мы используем для хранения знака.

Целочисленные типы данных

Рассмотрим числа, в двоичной записи которых не больше k цифр. Отведём место ровно под k цифр (битов).

Пример — типы языка C++ (`#include <cstdint>`):

Тип	Байты	Биты	Минимум	Максимум
`std::uint8_t`	1	8	0	255
`std::uint16_t`	2	16	0	65 535
`std::uint32_t`	4	32	0	$2^{32} - 1$
`std::uint64_t`	8	64	0	$2^{64} - 1$
`std::int8_t`	1	8	-128	127
`std::int16_t`	2	16	$-32\,768$	32 767
`std::int32_t`	4	32	-2^{31}	$2^{31} - 1$
`std::int64_t`	8	64	-2^{63}	$2^{63} - 1$

Для отрицательных чисел уменьшим диапазон на один бит. Этот бит мы используем для хранения знака.

Отрицательные числа

Если просто хранить знак, то:

- У нуля будет два различных представления ($+0$ и -0).
- При сложении и вычитании придётся разбирать случаи.

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

Код	Без знака	Со знаком
0000 0000	0	0
0000 0001	1	+1
0000 0010	2	+2
...	...	...
0111 1110	126	+126
0111 1111	127	+127
1000 0000	128	−128
1000 0001	129	−127
1000 0010	130	−126
...	...	...
1111 1110	254	−2
1111 1111	255	−1

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

Код	Без знака	Со знаком	Разность
0000 0000	0	0	0
0000 0001	1	+1	0
0000 0010	2	+2	0
...	...	...	...
0111 1110	126	+126	0
0111 1111	127	+127	0
1000 0000	128	−128	256
1000 0001	129	−127	256
1000 0010	130	−126	256
...	...	...	...
1111 1110	254	−2	256
1111 1111	255	−1	256

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

1. Интерпретация: арифметика по модулю 2^k.
   Хранится остаток числа по модулю 2^k. Можно интерпретировать его или как число без знака, или как число со знаком.

Пример: $1001\,0101_2 = 1 + 4 + 16 + 128 = 149$.

- Число без знака: какое число от 0 до 255 имеет остаток 149 от деления на $2^8 = 256$? Это число 149.
- Число со знаком: какое число от -128 до $+127$ имеет остаток 149 от деления на $2^8 = 256$? Это число -107.

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

2. Интерпретация: вычитание 2^k.
   Если число получается больше максимально возможного (то есть $2^{k-1} - 1$), отнимем от него 2^k.

Пример: $1001\,0101_2 = 1 + 4 + 16 + 128 = 149$.

$149 > 127$, значит, значение равно $149 - 256 = -107$.

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

3. Интерпретация: отрицательное основание для старшего бита.
   Обычно бит с номером i домножается на 2^i. А теперь старший бит (с номером $k-1$) домножается не на 2^{k-1}, а на -2^{k-1}.

Пример: $1001\,0101_2 = 1 + 4 + 16 + 128 = 149$.

Код $1001\,0101$ даёт $1 + 4 + 16 - 128 = -107$.

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

4. Интерпретация: механический способ.
   Чтобы получить из кода отрицательного числа его модуль, инвертируем все биты, а затем прибавим единицу.

Пример: $1001\,0101_2 = 1 + 4 + 16 + 128 = 149$.

$$\begin{aligned} \text{Инверсия } 1001\,0101_2 + 1 &= \\ 0110\,1010_2 + 1 &= \\ 0110\,1011_2 &= \\ 1 + 2 + 8 + 32 + 64 &= \\ 107 & \end{aligned}$$

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

Сложение и умножение делаются в столбик.

$$
\begin{aligned}
& 1001\,0101_2 && (= -107) \\
+\ & 0110\,1111_2 && (= +111) \\
= & \ 1\,0000\,0100_2 && (\text{оставим последние } k \text{ цифр результата}) \\
\to & \ 0000\,0100_2 && (= +4)
\end{aligned}
$$

Отрицательные числа

Двоичный дополнительный код — менее нагляден, но проще для механических вычислений.

Сложение и умножение делаются в столбик.

$$
\begin{aligned}
& 1111\,1011_2 && (= -5) \\
\times\ & 0000\,0111_2 && (= +7) \\
= & \ 1111\,1011_2 \\
+ & \ 1\,1111\,0110_2 \\
+ & \ 11\,1110\,1100_2 \\
= & \ 110\,1101\,1101_2 && \text{(оставим последние } k \text{ цифр результата)} \\
\to & \ 1101\,1101_2 && (= -35)
\end{aligned}
$$

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

1. Инвертирование всех битов.
2. Побитовая операция AND.
3. Побитовая операция OR.
4. Побитовая операция XOR.
5. Побитовый сдвиг влево.
6. Логический сдвиг вправо.
7. Арифметический сдвиг вправо.

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

1. Инвертирование всех битов.

$$\texttt{a} = 1001\,0011,$$
$$\texttt{\~{}a} = 0110\,1100.$$

A	~A
0	1
1	0

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

2. Побитовая операция AND.

$$\texttt{a} = 1001\,0011,$$
$$\texttt{b} = 1100\,1010,$$
$$\texttt{a \& b} = 1000\,0010.$$

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

3. Побитовая операция OR.

$$\texttt{a} = 1001\,0011,$$
$$\texttt{b} = 1100\,1010,$$
$$\texttt{a} \mid \texttt{b} = 1101\,1011.$$

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

4. Побитовая операция XOR.

$$\texttt{a} = 1001\,0011,$$
$$\texttt{b} = 1100\,1010,$$
$$\texttt{a ^ b} = 0101\,1001.$$

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

5. Побитовый сдвиг влево.

$$\begin{aligned} \texttt{a} &= 1001\,0011, \\ \texttt{b} &= 3, \\ \texttt{a} \ \texttt{<<}\ \texttt{b} &= 1001\,1000. \end{aligned}$$

Все биты сдвигаются влево `b` раз.
Старший бит пропадает.
Младший бит становится нулём.

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

6. Логический сдвиг вправо.

$$\begin{aligned} \texttt{a} &= 1001\,0011, \\ \texttt{b} &= 2, \\ \texttt{a} \;\texttt{>>>}\; \texttt{b} &= 0010\,0100. \end{aligned}$$

Все биты сдвигаются вправо `b` раз.
Младший бит пропадает.
Старший бит становится нулём.

Битовые операции

Представление целых чисел даёт возможность легко выполнять некоторые действия.

7. Арифметический сдвиг вправо.

$$\begin{aligned} \texttt{a} &= 1001\,0011, \\ \texttt{b} &= 2, \\ \texttt{a} \texttt{ >> } \texttt{b} &= 1110\,0100. \end{aligned}$$

Все биты сдвигаются вправо `b` раз.
Младший бит пропадает.
Старший (знаковый) бит не меняется.

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: 7654 3210

$$a = 1001\,0011$$

Операции с отдельными битами:

1. Как получить значение бита с номером k:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером k в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером k в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером k в значение v:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: 7654 3210

a = 1001 0011

$k = 3$

a >> k = 0001 0010

значение = 0000 0000

Операции с отдельными битами:

1. Как получить значение бита с номером k:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером k в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером k в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером k в значение v:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: 7654 3210 $\quad$ $k = 3$ $\quad$ `1 << k` $=$ 0000 1000

`a` $=$ 1001 0011 $\quad$ `a` $\leftarrow$ 1001 1011

Операции с отдельными битами:

1. Как получить значение бита с номером `k`:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером `k` в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером `k` в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером `k` в значение `v`:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: 7654 3210

$\texttt{a} = 1001\,0011$

$k = 3$

$\texttt{~(1 << k)} = 1111\,0111$

$\texttt{a} \leftarrow 1001\,0011$

Операции с отдельными битами:

1. Как получить значение бита с номером k:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером k в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером k в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером k в значение v:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: 7654 3210 $\quad k = 3 \quad$ v << k = 0000 v000

a = 1001 0011 $\quad$ a ← 1001 v011

Операции с отдельными битами:

1. Как получить значение бита с номером k:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером k в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером k в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером k в значение v:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Работа с отдельными битами

Биты нумеруются с нуля, начиная с младшего.
Номер бита равен степени двойки, на которую домножается этот бит.

номера битов: 7654 3210

a = 1001 0011

Операции с отдельными битами:

1. Как получить значение бита с номером `k`:
   `get (a, k): (a >> k) & 1`
2. Как установить бит с номером `k` в единицу:
   `set1 (a, k): a ← a | (1 << k)`
3. Как установить бит с номером `k` в ноль:
   `set0 (a, k): a ← a & ~(1 << k)`
4. Как установить бит с номером `k` в значение `v`:
   `set (a, k, v): a ← (a & ~(1 << k)) | (v << k)`

Содержание

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений.
- Удобство хранения.
- Удобство арифметических действий.

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений.
- Удобство хранения.
- Удобство арифметических действий.

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений: рациональные числа с небольшими числителем и знаменателем.
- Удобство хранения: да.
- Удобство арифметических действий: приведение к общему знаменателю: числитель и знаменатель быстро растут.

Способ 1 — рациональные числа.
Число представляется в виде $\frac{P}{Q}$, где P и Q — целые: $\frac{0}{1}$, $\frac{5}{3}$, $\frac{-6}{11}$, ...
Где применяется: точное решение некоторых математических задач.

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений: почти любое число хранится приближённо.
- Удобство хранения: да.
- Удобство арифметических действий: как с целыми.

Способ 2 — числа с фиксированной точкой.
Отдельно хранятся целая часть и дробная часть.
Например, $X = A.B = A + \frac{1}{65\,536}B$, где $0 \leqslant A, B < 65\,536$.
Можно считать, что хранятся целые числа, но их значения при использовании следует делить на 65 536.
Где применяется: Metafont и Metapost, графические библиотеки, Java BigDecimal.

Как хранить вещественное число?

Постановка задачи: требуется хранить вещественные числа и производить с ними вычисления.

Критерии качества:

- Диапазон значений: почти любое число хранится приближённо, можно хранить очень большие и очень маленькие по модулю числа.
- Удобство хранения: да.
- Удобство арифметических действий: почти как с целыми.

Способ 3 — числа с плавающей точкой.
Хранятся первые несколько двоичных знаков числа, а также степень двойки (небольшое положительное или отрицательное число), на которую нужно домножить.
Например, $X = 1.00101_2 \cdot 2^{100}$, $Y = -1 = -1.00000_2 \cdot 2^0$.
Та же конструкция с десятичными числами: $X = 1.23456 \cdot 10^{100}$, $Y = -1 = -1.00000 \cdot 10^0$.

Вещественные типы данных

Отведём место ровно под k первых двоичных цифр, ещё p битов на степень двойки и ещё один бит на знак числа.

Пример — типы компилятора GNU C++ для архитектуры x86:

Вещественные типы данных

Отведём место ровно под k первых двоичных цифр, ещё p битов на степень двойки и ещё один бит на знак числа.

Пример — типы компилятора GNU C++ для архитектуры x86:

Тип	**Байты**	**s**	**p**	**k**	**Минимум**	**Максимум**
`float`	4	1	8	23	$\pm 1.5 \cdot 10^{-45}$	$\pm 3.4 \cdot 10^{+38}$
`double`	8	1	11	52	$\pm 5.0 \cdot 10^{-324}$	$\pm 1.7 \cdot 10^{+308}$
`long double`	10	1	15	64	$\pm 3.4 \cdot 10^{-4932}$	$\pm 1.1 \cdot 10^{+4932}$

Типы `float` и `double` реализуют стандарт IEEE-754, а тип `long double` — его расширение.

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.

Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 1: число
$1 = (-1)^0 \times 1.0000\ldots00 \times 2^{127-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1111 1000 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.

Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 1: число

$1 = (-1)^0 \times 1.0000\ldots00 \times 2^{127-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1111 1000 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 1: число
$1 = (-1)^0 \times 1.0000\ldots00 \times 2^{127-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1111 1000 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 1: число
$1 = (-1)^0 \times 1.0000\ldots00 \times 2^{127-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1111 1000 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 2: число
$-5.5 = (-1)^1 \times 1.0110\ldots00 \times 2^{129-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
1100 0000 1011 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 2: число
$-5.5 = (-1)^1 \times 1.0110\ldots00 \times 2^{129-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
1100 0000 1011 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 2: число
$-5.5 = (-1)^1 \times 1.0110\ldots00 \times 2^{129-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
1100 0000 1011 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.
Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 2: число
$-5.5 = (-1)^1 \times 1.0110\ldots00 \times 2^{129-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
1100 0000 1011 0000 0000 0000 0000 0000
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.

Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 3: число
$\frac{1}{5} = (-1)^0 \times 1.10011001100110011001100110011\ldots \times 2^{124-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1110 0100 1100 1100 1100 1100 1101
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.

Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 3: число

$\frac{1}{5} = (-1)^0 \times 1.10011001100110011001100110011\ldots \times 2^{124-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1110 0100 1100 1100 1100 1100 1101
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.

Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 3: число

$\frac{1}{5} = (-1)^0 \times 1.10011001100110011001100110011\ldots \times 2^{124-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
00011 1110 00100 1100 1100 1100 1100 1101
```

Числа с плавающей точкой

Представление числа:

$$X = (-1)^S \times 1.M \times 2^{E-E_0}$$

Значение $E_0 = 2^{p-1} - 1$.

Мы будем рассматривать примеры для типа `float` ($E_0 = 127$).

Пример 3: число

$\frac{1}{5} = (-1)^0 \times 1.10011001100110011001100110011\ldots \times 2^{124-127}$.

```
seee eeee emmm mmmm mmmm mmmm mmmm mmmm
|[-- ---- ][-- ---- ---- ---- ---- ----]
0011 1110 0100 1100 1100 1100 1100 1101
```

Числа с плавающей точкой

Как получается двоичная дробь для $\frac{1}{5} = 0.0011\ldots$?

- $\frac{1}{5} = 0 \cdot 1 + \frac{2}{5} \cdot \frac{1}{2}$
- $\frac{2}{5} \cdot \frac{1}{2} = 0 \cdot \frac{1}{2} + \frac{4}{5} \cdot \frac{1}{4}$
- $\frac{4}{5} \cdot \frac{1}{4} = 0 \cdot \frac{1}{4} + \frac{8}{5} \cdot \frac{1}{8}$
- $\frac{8}{5} \cdot \frac{1}{8} = 1 \cdot \frac{1}{8} + \frac{3}{5} \cdot \frac{1}{8} = 1 \cdot \frac{1}{8} + \frac{6}{5} \cdot \frac{1}{16}$
- $\frac{6}{5} \cdot \frac{1}{16} = 1 \cdot \frac{1}{16} + \frac{1}{5} \cdot \frac{1}{16} = 1 \cdot \frac{1}{16} + \frac{2}{5} \cdot \frac{1}{32}$
- $\ldots$

Специальные значения

Кроме чисел, в типе с плавающей точкой могут храниться специальные значения.

Знак	Экспонента	Мантисса	Значение
0	00..00	00..00	$+0$
0	00..00	00..01 ...11..11	$+0.m \times 2^{-e_0+1}$
0	00..01 ...11..10	XX..XX	$+1.m \times 2^{e-e_0}$
0	11..11	00..00	$+\infty$
0	11..11	00..01 ...01..11	SNaN
0	11..11	10..00 ...11..11	QNaN
1	00..00	00..00	-0
1	00..00	00..01 ...11..11	$-0.m \times 2^{-e_0+1}$
1	00..01 ...11..10	XX..XX	$-1.m \times 2^{e-e_0}$
1	11..11	00..00	$-\infty$
1	11..11	00..01 ...01..11	SNaN
1	11..11	10..00 ...11..11	QNaN

Специальные значения

Кроме чисел, в типе с плавающей точкой могут храниться специальные значения.

Знак	Экспонента	Мантисса	Значение
0	00..00	00..00	$+0$
0	00..00	00..01 ...11..11	$+0.m \times 2^{-e_0+1}$
0	00..01 ...11..10	XX..XX	$+1.m \times 2^{e-e_0}$
0	11..11	00..00	$+\infty$
0	11..11	00..01 ...01..11	SNaN
0	11..11	10..00 ...11..11	QNaN
1	00..00	00..00	-0
1	00..00	00..01 ...11..11	$-0.m \times 2^{-e_0+1}$
1	00..01 ...11..10	XX..XX	$-1.m \times 2^{e-e_0}$
1	11..11	00..00	$-\infty$
1	11..11	00..01 ...01..11	SNaN
1	11..11	10..00 ...11..11	QNaN

Содержание

Сложение short, пример 1

Сложение short, пример 1: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile short a;
4      for (a = 0; a >= 0; a++)
5          ;
6      printf ("%d\n", a);
7      return 0;
8  }
```

Сложение short, пример 1

Сложение short, пример 1: `-32768`.

```
1  #include <stdio.h>
2  int main () {
3      volatile short a;
4      for (a = 0; a >= 0; a++)
5          ;
6      printf ("%d\n", a);
7      return 0;
8  }
```

Переполнение: `32767 + 1 = -32768`.

Вообще это неопределённое поведение (undefined behavior).

Сложение short, пример 2

Сложение short, пример 2: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile unsigned short a;
4      for (a = 0; a >= 0; a++)
5          ;
6      printf ("%d\n", a);
7      return 0;
8  }
```

Сложение short, пример 2

Сложение short, пример 2: *(вечный цикл).*

```
1  #include <stdio.h>
2  int main () {
3      volatile unsigned short a;
4      for (a = 0; a >= 0; a++)
5          ;
6      printf ("%d\n", a);
7      return 0;
8  }
```

Все числа типа `unsigned short` неотрицательные.

Переполнение `unsigned`-типов определено однозначно.

Сложение short, пример 3

Сложение short, пример 3: что выведет программа?

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  int main () {
 4      volatile short a, b, c, d;
 5      srand (0);
 6      while (1) {
 7          a = rand ();
 8          b = rand ();
 9          c = a + 5;
10          d = b + 5;
11          if ((a <= b) != (c <= d))
12              break;
13      }
14      printf ("%d %d %d %d\n", a, b, c, d);
15      return 0;
16  }
```

Сложение short, пример 3

Сложение short, пример 3: `10731 32766 10736 -32765`.

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  int main () {
 4      volatile short a, b, c, d;
 5      srand (0);
 6      while (1) {
 7          a = rand ();
 8          b = rand ();
 9          c = a + 5;
10          d = b + 5;
11          if ((a <= b) != (c <= d))
12              break;
13      }
14      printf ("%d %d %d %d\n", a, b, c, d);
15      return 0;
16  }
```

Одна из сумм переполнилась, а другая нет.

Вообще это неопределённое поведение (undefined behavior).

Сложение short, пример 4

Сложение short, пример 4: что выведет программа?

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  int main () {
 4      volatile unsigned short a, b, c, d;
 5      srand (0);
 6      while (1) {
 7          a = rand () * 2;
 8          b = rand () * 2;
 9          c = a + 5;
10          d = b + 5;
11          if ((a <= b) != (c <= d))
12              break;
13      }
14      printf ("%d %d %d %d\n", a, b, c, d);
15      return 0;
16  }
```

Сложение short, пример 4

Сложение short, пример 4: `21462 65532 21467 1`.

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  int main () {
 4      volatile unsigned short a, b, c, d;
 5      srand (0);
 6      while (1) {
 7          a = rand () * 2;
 8          b = rand () * 2;
 9          c = a + 5;
10          d = b + 5;
11          if ((a <= b) != (c <= d))
12              break;
13      }
14      printf ("%d %d %d %d\n", a, b, c, d);
15      return 0;
16  }
```

Одна из сумм переполнилась, а другая нет.

Переполнение `unsigned`-типов определено однозначно.

Вывод байтов из int, пример

Вывод байтов из int, пример: что выведет программа?

```
 1  #include <stdio.h>
 2  union share {
 3      int as_int;
 4      unsigned char as_arr [4];
 5  };
 6  int main () {
 7      volatile share s;
 8      s.as_int = 12345;
 9      for (int i = 3; i >= 0; i--)
10          printf ("%02X ", s.as_arr[i]);
11      return 0;
12  }
```

Вывод байтов из int, пример

Вывод байтов из int, пример: `00 00 30 39` .

```
1  #include <stdio.h>
2  union share {
3      int as_int;
4      unsigned char as_arr [4];
5  };
6  int main () {
7      volatile share s;
8      s.as_int = 12345;
9      for (int i = 3; i >= 0; i--)
10         printf ("%02X ", s.as_arr[i]);
11     return 0;
12 }
```

Выведутся отдельные байты из `int`, начиная со старшего.

Вывод байтов из float, пример

Вывод байтов из float, пример: что выведет программа?

```
 1  #include <stdio.h>
 2  union share {
 3      float as_float;
 4      unsigned char as_arr [4];
 5  };
 6  int main () {
 7      volatile share s;
 8      s.as_float = 1.0 / 5.0;
 9      for (int i = 3; i >= 0; i--)
10          printf ("%02X ", s.as_arr[i]);
11      return 0;
12  }
```

Вывод байтов из float, пример

Вывод байтов из float, пример: `3E 4C CC CD` .

```
1  #include <stdio.h>
2  union share {
3      float as_float;
4      unsigned char as_arr [4];
5  };
6  int main () {
7      volatile share s;
8      s.as_float = 1.0 / 5.0;
9      for (int i = 3; i >= 0; i--)
10         printf ("%02X ", s.as_arr[i]);
11     return 0;
12 }
```

Выведутся отдельные байты из `float`, начиная со старшего.

Сложение float, пример 1

Сложение float, пример 1: что выведет программа?

```
 1  #include <stdio.h>
 2  int main () {
 3      volatile float a, b;
 4      a = 0;
 5      while (1) {
 6          b = a + 1;
 7          if (a == b)
 8              break;
 9          a = b;
10      }
11      printf ("%f\n", a);
12      return 0;
13  }
```

Сложение float, пример 1

Сложение float, пример 1: 16777216.000000.

```
 1  #include <stdio.h>
 2  int main () {
 3      volatile float a, b;
 4      a = 0;
 5      while (1) {
 6          b = a + 1;
 7          if (a == b)
 8              break;
 9          a = b;
10      }
11      printf ("%f\n", a);
12      return 0;
13  }
```

Программа выведет число $16777216.000000 = 2^{24}$.

Сложение float, пример 2

Сложение float, пример 2: что выведет программа?

```
 1  #include <stdio.h>
 2  int main () {
 3      volatile float a, b;
 4      a = 1E10;
 5      b = 1E10 + 1000;
 6      while (a < b)
 7          a = a + 1;
 8      printf ("%f\n", a);
 9      return 0;
10  }
```

Сложение float, пример 2

Сложение float, пример 2: *(вечный цикл).*

```
 1  #include <stdio.h>
 2  int main () {
 3      volatile float a, b;
 4      a = 1E10;
 5      b = 1E10 + 1000;
 6      while (a < b)
 7          a = a + 1;
 8      printf ("%f\n", a);
 9      return 0;
10  }
```

Это вечный цикл: увеличение a на единицу фактически не происходит.

Сложение float, пример 3

Сложение float, пример 3: что выведет программа?

```
 1  #include <stdio.h>
 2  int main () {
 3      volatile float a, b;
 4      a = 1E10;
 5      b = 1E10 + 10;
 6      while (a < b)
 7          a = a + 1;
 8      printf ("%f\n", a);
 9      return 0;
10  }
```

Сложение float, пример 3

Сложение float, пример 3: 10000000000.000000.

```
 1  #include <stdio.h>
 2  int main () {
 3      volatile float a, b;
 4      a = 1E10;
 5      b = 1E10 + 10;
 6      while (a < b)
 7          a = a + 1;
 8      printf ("%f\n", a);
 9      return 0;
10  }
```

Этот цикл не выполнится ни разу, так как a = b.

Будет выведено число 10000000000.000000.

Сложение float, пример 4

Сложение float, пример 4: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile float a;
4      a = 1;
5      a += 1E10;
6      a -= 1E10;
7      printf ("%f\n", a);
8      return 0;
9  }
```

Сложение float, пример 4

Сложение float, пример 4: `0.000000`.

```
1  #include <stdio.h>
2  int main () {
3      volatile float a;
4      a = 1;
5      a += 1E10;
6      a -= 1E10;
7      printf ("%f\n", a);
8      return 0;
9  }
```

После сложения на хранение единицы не хватит разрядов.
Будет выведено число `0.000000`.

Сложение float, пример 5

Сложение float, пример 5: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile float a;
4      a = 1000;
5      a += 1E10;
6      a -= 1E10;
7      printf ("%f\n", a);
8      return 0;
9  }
```

Сложение float, пример 5

Сложение float, пример 5: 1024.000000.

```
1  #include <stdio.h>
2  int main () {
3      volatile float a;
4      a = 1000;
5      a += 1E10;
6      a -= 1E10;
7      printf ("%f\n", a);
8      return 0;
9  }
```

После сложения не все разряды тысячи удастся сохранить.
Будет выведено число 1024.000000.

Сложение float, пример 6

Сложение float, пример 6: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile float a, b;
4      a = 16777216;
5      b = 1;
6      a += b;
7      printf ("%f\n", a);
8      return 0;
9  }
```

Сложение float, пример 6

Сложение float, пример 6: `16777216.000000`.

```c
#include <stdio.h>
int main () {
    volatile float a, b;
    a = 16777216;
    b = 1;
    a += b;
    printf ("%f\n", a);
    return 0;
}
```

```
             987654321098765432109876543210
                    =======================
2^{23}     =       100000000000000000000000
2^{23} + 1 =       100000000000000000000001
                    =======================
             987654321098765432109876543210
```

Сложение float, пример 6

Сложение float, пример 6: 16777216.000000.

```
1  #include <stdio.h>
2  int main () {
3      volatile float a, b;
4      a = 16777216;
5      b = 1;
6      a += b;
7      printf ("%f\n", a);
8      return 0;
9  }
```

```
                  98765432109876543210987654321O
                       ==========================
2^{24}       =       100000000000000000000000.0
2^{24} + 1   =       100000000000000000000000.1
                       ==========================
                  98765432109876543210987654321O
```

Сложение float, пример 7

Сложение float, пример 7: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile float a, b;
4      a = 16777218;
5      b = 1;
6      a += b;
7      printf ("%f\n", a);
8      return 0;
9  }
```

Сложение float, пример 7

Сложение float, пример 7: 16777220.000000.

```
1  #include <stdio.h>
2  int main () {
3      volatile float a, b;
4      a = 16777218;
5      b = 1;
6      a += b;
7      printf ("%f\n", a);
8      return 0;
9  }
```

```
                  98765432109876543210987654321O
                       ========================
2^{24} + 2 =           1000000000000000000000010
2^{24} + 3 =           1000000000000000000000011
2^{24} + 4 =           1000000000000000000000100
                       ========================
                  98765432109876543210987654321O
```

Сложение float, пример 8

Сложение float, пример 8: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile float a, b;
4      a = 33554432;
5      b = 2;
6      a = a + b;
7      printf ("%f\n", a);
8      return 0;
9  }
```

Сложение float, пример 8

Сложение float, пример 8: 33554432.000000.

```
1  #include <stdio.h>
2  int main () {
3      volatile float a, b;
4      a = 33554432;
5      b = 2;
6      a = a + b;
7      printf ("%f\n", a);
8      return 0;
9  }
```

```
                  98765432109876543210987654321O
                      ==========================
2^{25}          =     1000000000000000000000000
2^{25} + 2      =     1000000000000000000000010
                      ==========================
                  98765432109876543210987654321O
```

Сложение float, пример 9

Сложение float, пример 9: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile float a, b;
4      a = 33554432;
5      b = 3;
6      a = a + b;
7      printf ("%f\n", a);
8      return 0;
9  }
```

Сложение float, пример 9

Сложение float, пример 9: 33554436.000000.

```
1  #include <stdio.h>
2  int main () {
3      volatile float a, b;
4      a = 33554432;
5      b = 3;
6      a = a + b;
7      printf ("%f\n", a);
8      return 0;
9  }
```

```
                  98765432109876543210987654321O
                      ========================
2^{25}        =       10000000000000000000000000
2^{25} + 3    =       10000000000000000000000011
2^{25} + 4    =       10000000000000000000000100
                      ========================
                  98765432109876543210987654321O
```

Цикл по float, пример

Цикл по float, пример: что выведет программа?

```
1  #include <stdio.h>
2  int main () {
3      volatile float a;
4      for (a = 0.0; a != 1.0; a += 0.1)
5          ;
6      printf ("%f\n", a);
7      return 0;
8  }
```

Цикл по float, пример

Цикл по float, пример: *(вечный цикл).*

```
1  #include <stdio.h>
2  int main () {
3      volatile float a;
4      for (a = 0.0; a != 1.0; a += 0.1)
5          ;
6      printf ("%f\n", a);
7      return 0;
8  }
```

Число 0.1 не хранится точно.

Сумма десяти таких чисел не оказалась в точности равна 1.0.

Вычитание float, пример 1

Вычитание float, пример 1: что выведет программа?

```
1   #include <stdio.h>
2   #include <math.h>
3   int main () {
4       volatile float a, b, c, s, t;
5       a = 1.0 / 5.0;
6       b = 1.0 / 15.0;
7       c = b; c += b; c += b;
8       t = a - c;
9       s = sqrt (t);
10      printf ("%.9f %.9f %.9f\n%.9f %.9f\n", a, b, c, t, s);
11      return 0;
12  }
```

Вычитание float, пример 1

Вычитание float, пример 1: *(не получилось посчитать).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, s, t;
 5      a = 1.0 / 5.0;
 6      b = 1.0 / 15.0;
 7      c = b; c += b; c += b;
 8      t = a - c;
 9      s = sqrt (t);
10      printf ("%.9f %.9f %.9f\n%.9f %.9f\n", a, b, c, t, s);
11      return 0;
12  }
```

```
0.200000003 0.066666670 0.200000018
-0.000000015 -1.#IND00000
```

Вычитание float, пример 1

Вычитание float, пример 1: *(не получилось посчитать).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, s, t;
 5      a = 1.0 / 5.0;
 6      b = 1.0 / 15.0;
 7      c = b; c += b; c += b;
 8      t = a - c;
 9      s = sqrt (t);
10      printf ("%.9f %.9f %.9f\n%.9f %.9f\n", a, b, c, t, s);
11      return 0;
12  }
```

```
0.200000003 0.066666670 0.200000018
-0.000000015 -1.#IND00000
```

Числа $\frac{1}{5}$ и $\frac{1}{15}$ не хранятся точно.

Разность оказалась отрицательной.

Корень вычислить не удалось.

Вычитание float, пример 2

Вычитание float, пример 2: что выведет программа?

```
1   #include <stdio.h>
2   #include <math.h>
3   int main () {
4       volatile float a, b, c, s, t;
5       a = 1.0 / 5.0;
6       b = 1.0 / 25.0;
7       c = b; c += b; c += b; c += b; c += b;
8       t = a - c;
9       s = sqrt (t);
10      printf ("%.9f %.9f %.9f\n%.9f %.9f\n", a, b, c, t, s);
11      return 0;
12  }
```

Вычитание float, пример 2

Вычитание float, пример 2: *(ошибка в четвёртом знаке).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, s, t;
 5      a = 1.0 / 5.0;
 6      b = 1.0 / 25.0;
 7      c = b; c += b; c += b; c += b; c += b;
 8      t = a - c;
 9      s = sqrt (t);
10      printf ("%.9f %.9f %.9f\n%.9f %.9f\n", a, b, c, t, s);
11      return 0;
12  }
```

```
0.200000003 0.039999999 0.199999988
0.000000015 0.000122070
```

Вычитание float, пример 2

Вычитание float, пример 2: *(ошибка в четвёртом знаке).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, s, t;
 5      a = 1.0 / 5.0;
 6      b = 1.0 / 25.0;
 7      c = b; c += b; c += b; c += b; c += b;
 8      t = a - c;
 9      s = sqrt (t);
10      printf ("%.9f %.9f %.9f\n%.9f %.9f\n", a, b, c, t, s);
11      return 0;
12  }
```

```
0.200000003 0.039999999 0.199999988
0.000000015 0.000122070
```

Числа $\frac{1}{5}$ и $\frac{1}{25}$ не хранятся точно.

В разности ошибка в восьмом знаке после десятичной точки.

А в корне — уже в четвёртом знаке.

Вычитание float, пример 3

Вычитание float, пример 3: что выведет программа?

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 2.0; y2 = 3.0;
 7      x3 = 3.0; y3 = 5.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

Вычитание float, пример 3

Вычитание float, пример 3: *(ошибка в пятом знаке).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 2.0; y2 = 3.0;
 7      x3 = 3.0; y3 = 5.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

```
3.605551243 2.236068010 5.830951691 5.836285591
0.500010371
```

Вычитание float, пример 3

Вычитание float, пример 3: *(ошибка в пятом знаке).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 2.0; y2 = 3.0;
 7      x3 = 3.0; y3 = 5.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

```
3.605551243 2.236068010 5.830951691 5.836285591
0.500010371
```

Полупериметр близок к длине большей стороны.

Ошибка в пятом знаке.

Вычитание float, пример 4

Вычитание float, пример 4: что выведет программа?

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 21.0; y2 = 34.0;
 7      x3 = 34.0; y3 = 55.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

Вычитание float, пример 4

Вычитание float, пример 4: *(ошибка в первом знаке).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 21.0; y2 = 34.0;
 7      x3 = 34.0; y3 = 55.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

```
39.962482452 24.698177338 64.660652161 64.660659790
0.697788477
```

Вычитание float, пример 4

Вычитание float, пример 4: *(ошибка в первом знаке).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 21.0; y2 = 34.0;
 7      x3 = 34.0; y3 = 55.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

```
39.962482452 24.698177338 64.660652161 64.660659790
0.697788477
```

Полупериметр близок к длине большей стороны.

Ошибка в первом знаке.

Вычитание float, пример 5

Вычитание float, пример 5: что выведет программа?

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 34.0; y2 = 55.0;
 7      x3 = 55.0; y3 = 89.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

Вычитание float, пример 5

Вычитание float, пример 5: *(числа получились равными).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 34.0; y2 = 55.0;
 7      x3 = 55.0; y3 = 89.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

```
64.660652161 39.962482452 104.623130798 104.623130798
0.000000000
```

Вычитание float, пример 5

Вычитание float, пример 5: *(числа получились равными).*

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  int main () {
 4      volatile float a, b, c, p, s, x1, x2, x3, y1, y2, y3;
 5      x1 = 0.0; y1 = 0.0;
 6      x2 = 34.0; y2 = 55.0;
 7      x3 = 55.0; y3 = 89.0;
 8      a = hypot (y2 - y1, x2 - x1);
 9      b = hypot (y3 - y2, x3 - x2);
10      c = hypot (y1 - y3, x1 - x3);
11      p = (a + b + c) * 0.5;
12      s = p * (p - a) * (p - b) * (p - c);
13      s = sqrt (s);
14      printf ("%.9f %.9f %.9f %.9f\n%.9f\n", a, b, c, p, s);
15      return 0;
16  }
```

```
64.660652161 39.962482452 104.623130798 104.623130798
0.000000000
```

Полупериметр близок к длине большей стороны.
Настолько, что они получились равными.

Вопросы?

Вопросы?