

# **Курс «Алгоритмы и алгоритмические языки»**

## **Лекция 9**

## Рекурсия

- ◇ В языке Си функция может быть *рекурсивной*, т.е. вызывать саму себя:

```
int fibrec (int n)
{
    if (n == 1 || n == 2)
        return 1;
    else
        return fibrec (n - 2) + fibrec (n - 1);
}
```

- ◇ Рекурсивные функции часто неэффективны по сравнению с их нерекурсивными вариантами:

```
int fbn (int n)
{
    int i, g, h, fb;
    if (n == 1 || n == 2)
        return 1;
    else
        for (i = 2, g = h = 1; i < n; i++) {
            fb = g + h;
            h = g;
            g = fb;
        }
    return fb;
}
```

## Ключевое слово `inline`: подставляемые функции (C99)

```
#include <stdio.h>
inline static int max (int a, int b)
{
    return a > b ? a : b;
}
int main (void)
{
    int x = 5, y = 17;
    printf ("Наибольшим из чисел %d и %d является %d\n",
           x, y, max (x, y));
    return 0;
}
```

◇ При обычной реализации `inline` приведенная программа эквивалентна:

```
#include <stdio.h>
inline static int max (int a, int b)
{
    return a > b ? a : b;
}
int main (void)
{
    int x = 5, y = 17;
    printf ("Наибольшим из чисел %d и %d является %d\n",
           x, y, (x > y ? x : y));
    return 0;
}
```

## Операция `sizeof`

- ◆ Одноместная операция `sizeof` позволяет определить длину операнда в байтах.
  - Операнды – типы либо переменные.
  - Результат имеет тип `size_t`
- ◆ Операция `sizeof` выполняется во время компиляции, ее результат представляет собой константу.
- ◆ Операция `sizeof` помогает улучшить переносимость программ.
- ◆ Операция `sizeof` позволяет определить, например, объем памяти в байтах, занимаемый двумерным массивом:  
`number_of_bytes = d1 × d2 × sizeof (element_type)`  
где `d1` – количество элементов по первому измерению,  
`d2` – количество элементов по второму измерению,  
`element_type` – тип элемента массива.
- ◆ Можно поступить и проще:  
`number_of_bytes = sizeof (имя_массива)`

## Операция sizeof

◇ Пример:

```
#include <stdio.h>
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[10];

    /* копирование 9 символов из argv[1] в buffer;
       sizeof (char) равно 1, число элементов массива
       buffer равно его размеру в байтах */
    strncpy (buffer, argv[1],
             sizeof (buffer) - sizeof (char));
    buffer[sizeof (buffer) - 1] = '\0';
    return 0;
}
```

## Операция `sizeof`

- ◆ `sizeof` можно применять только к «полностью» определенным типам.

Для массивов это означает:

- ◆ размерности массива должны присутствовать в его объявлении
- ◆ тип элементов массива должен быть полностью определен.

- ◆ Пример. Если объявление массива имеет вид:

```
extern int arr[];
```

то операция `sizeof (arr)` ошибочна, так как у компилятора нет возможности узнать, сколько элементов содержит массив `arr`.

## **Указатели на функцию**

- ◇ Каждая функция располагается в памяти по определенному адресу. Адресом функции является ее точка входа (при вызове функции управление передается именно на эту точку).
- ◇ Присвоив значение адреса функции переменной типа указатель, получим указатель на функцию.
- ◇ Указатель функции можно использовать вместо ее имени при вызове этой функции. Указатель «лучше» имени тем, что его можно передавать другим функциям в качестве их аргумента.
- ◇ Имя функции `f ( )` без скобок и аргументов (`f`) по определению является указателем на функцию `f ( )` (аналогия с массивом).

## Указатели на функцию

- ◆ **Пример.** Сравнение двух строк символов, введенных пользователем (функция `check()`).

```
#include <stdio.h>
#include <string.h>
typedef int (*pfunc_t) (const char*, const char*);

static void check (char *a, char *b, pfunc_t pf) {
    printf ("Проверка на совпадение: ");
    if (! pf (a, b))
        printf ("равны\n");
    else
        printf ("не равны\n");
}

int main (void) {
    char s1[80], s2[80];

    printf ("Введите две строки \n");
    fgets (s1, sizeof (s1), stdin); s1[strlen (s1) - 1] = 0;
    fgets (s2, sizeof (s2), stdin); s2[strlen (s2) - 1] = 0;
    check (s1, s2, strcmp);
    return 0;
}
```



- ◆ Объявление `int (*p)(const char *, const char *);` сообщает компилятору, что `p` – указатель на функцию, имеющую два параметра типа `const char *` и возвращающую значение типа `int`.
- ◆ Скобки вокруг `*p` нужны, так как операция `*` имеет более низкий приоритет, чем `()`: Если написать `int *p(...)`, получится, что объявлен не указатель на функцию, а функция `p`, которая возвращает указатель на целое.
- ◆ `(*cmp)(a, b)` эквивалентно `cmp(a, b)`.
- ◆ У функции `check` три параметра: два указателя на тип `char` и указатель на функцию `pf`. Указатель `pf` и функция `strcmp` имеют одинаковый формат, что позволяет использовать имя функции в качестве аргумента, соответствующего параметру `pf`.
- ◆ В данном случае использование указателя функции позволяет не менять программу сравнения, и тем самым получается более общий алгоритм.

```
int compvalues (const char *a, const char *b) {  
    return atoi (a) != atoi (b);  
}
```

# Вычисления с плавающей точкой

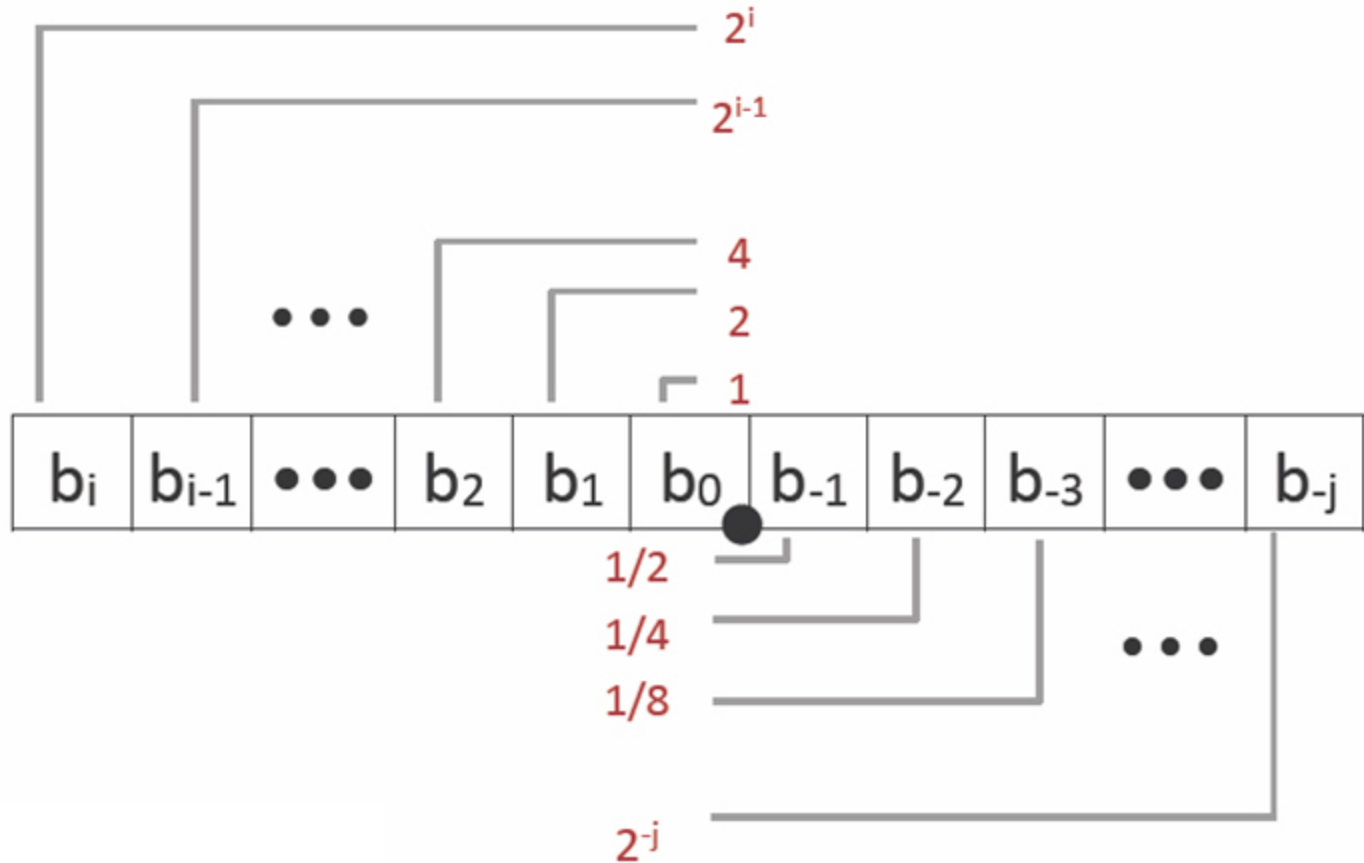
- ◇ Предпосылки: дробные двоичные числа
- ◇ Стандарт арифметики с плавающей точкой IEEE 754:  
Определение
- ◇ Пример и свойства
- ◇ Округление, сложение, умножение
- ◇ Плавающие типы языка Си
- ◇ Выводы

## Дробные двоичные числа

◇ Что такое  $1011.101_2$  ?

$$\begin{aligned} & 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = \\ & = 11 \frac{5}{8} = 11.625 \end{aligned}$$

# Дробные двоичные числа



- ◆ Черное пятнышко – двоичная точка
- ◆ Биты слева от точки умножаются на положительные степени 2
- ◆ Биты справа от точки умножаются на отрицательные степени 2

## Дробные двоичные числа

◇  $0.111111\dots_2 = 1.0 - \varepsilon$  ( $\varepsilon \rightarrow 0$ ), так как

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^n} + \dots \rightarrow 1 \quad \text{при } n \rightarrow \infty$$

◇ Точно можно представить только числа вида  $\frac{x}{2^k}$

◇ Остальные рациональные числа представляются периодическими двоичными дробями:

$$\frac{1}{5} = 0.(0011)_2$$

◇ Иррациональные числа представляются аperiodическими двоичными дробями и могут быть представлены только приближенно

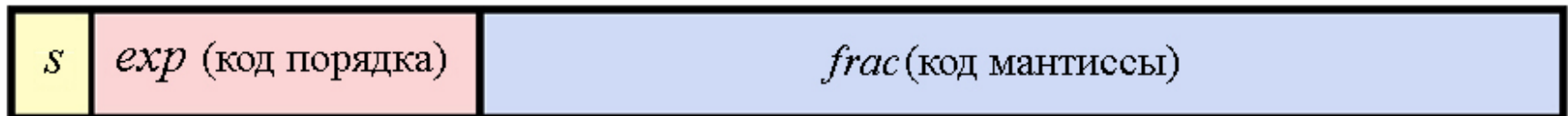
# Представление чисел с плавающей точкой (IEEE 754)

- ◆ Числа с плавающей точкой представляются в нормализованной форме:  $(-1^s) M 2^e$ 
  - ◆  $s$  – код знака числа (он же знак мантиссы)
  - ◆  $M$  – мантисса (  $1 \leq M < 2$  )
  - ◆  $e$  – (двоичный) порядок
  
- ◆ Первая цифра мантиссы в нормализованном представлении всегда 1. В стандарте принято решение не записывать в представлении числа эту единицу (тем самым мантисса как бы увеличивается на разряд).

Экономия связана с тем, что в представление числа записывается не  $M$ , а  $frac = M - 1$

## Представление чисел с плавающей точкой

- ◆ Чтобы не записывать отрицательных чисел в поле порядка, вводится *смещение*  $bias = 2^{k-1} - 1$ , где  $k$  – количество бит в поле для записи порядка, и вместо порядка  $e$  записывается код порядка  $exp$ , связанный с  $e$  соотношением  $e = exp - bias$ .
- ◆ Нормализованное число  $(-1^s) M 2^e$  упаковывается в машинное слово (структуру) с полями  $s$ ,  $frac$  и  $exp$

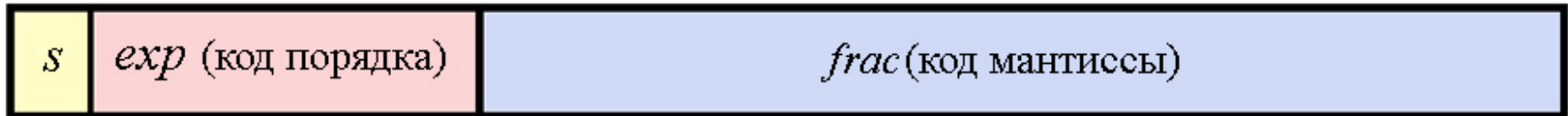


Ширина поля  $s$  всегда равна 1.

Ширина полей  $exp$  и  $frac$  зависит от точности числа

# Представление чисел с плавающей точкой

◇ Одинарная точность (32 бита):

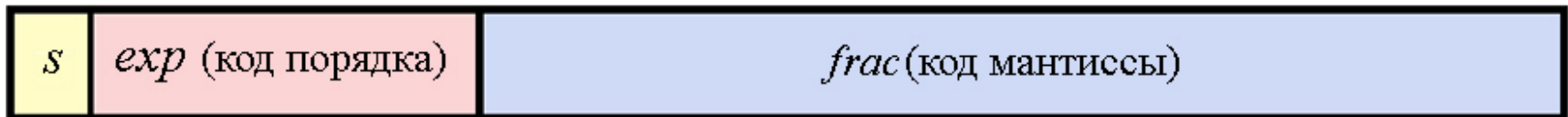


8 бит

23 бита

$$bias = 127; \quad -126 \leq e \leq 127; \quad 1 \leq exp \leq 254$$

◇ Двойная точность (64 бита):

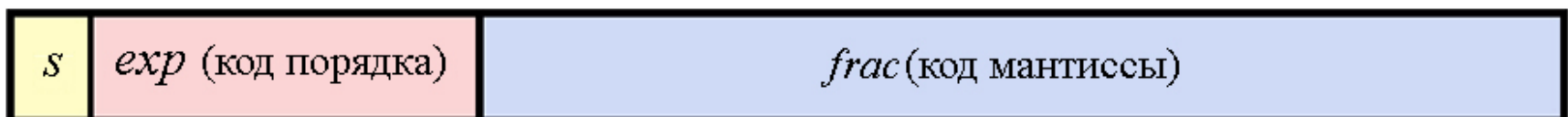


11 бит

52 бита

$$bias = 1023; \quad -1022 \leq e \leq 1023; \quad 1 \leq exp \leq 2046$$

◇ Повышенная точность (80 бит):



15 бит

64 бита



# Представление чисел с плавающей точкой

## ◆ Пример

◆ Значение float  $f = 15213.0$

$$15213_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$$

◆ Значащая часть

$$M = 1.\underline{1101101101101}_2,$$

$$\text{frac} = \underline{110110110110100000000000}_2$$

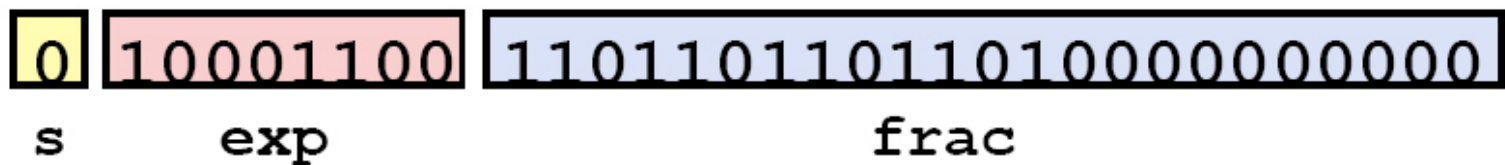
◆ Порядок

$$e = 13$$

$$\text{bias} = 127$$

$$\text{exp} = 140 = 10001100_2$$

◆ Результат



## Представление нуля

- ◆ Для типа `float` код порядка `exp` изменяется от `00000001` до `11111111`  
(значению `00000001` соответствует порядок  $e = -126$ ,  
значению `11111111` – порядок  $e = 127$ )
- ◆ Код `exp = 00000000, frac = 000...0`  
представляет нулевое значение; в зависимости от значения знакового разряда `s` это либо `+0` либо `-0`
- ◆ А какое значение представляет код  
`exp = 00000000, frac ≠ 000...0?`

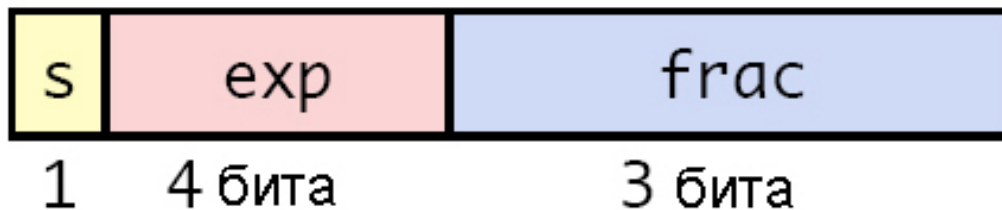
## Большие числа

Пусть  $\text{exp} = 111\dots 1$

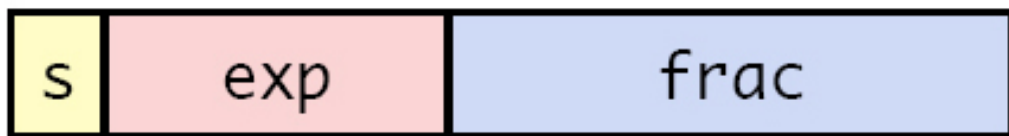
- ◇ если при этом  $\text{frac} = 000\dots 0$ , то коду будет соответствовать значение  $\infty$  (со знаком  $s$ )
- ◇ если же  $\text{frac} \neq 000\dots 0$ , то код не будет представлять никакое число («значение», представляемое таким кодом, так и называется: «не число» – NaN – Not a number)

## Денормализованные числа

- ◇ Это числа, представляемые кодами  
 $\text{exp} = 00000000, \text{frac} \neq 000\dots0$
- ◇  $\text{exp}$  вносит в значение такого числа постоянный вклад  $2^{-k-2}$ ,  
 $\text{frac}$  меняется от  $000\dots01$  до  $111\dots1$  и рассматривается уже не как мантисса, а как значение, умножаемое на  $\text{exp}$
- ◇ Рассмотрим это на модельном примере:



# 8-разрядные числа с плавающей точкой (положительные)

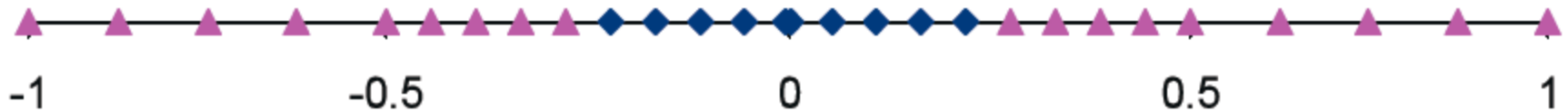
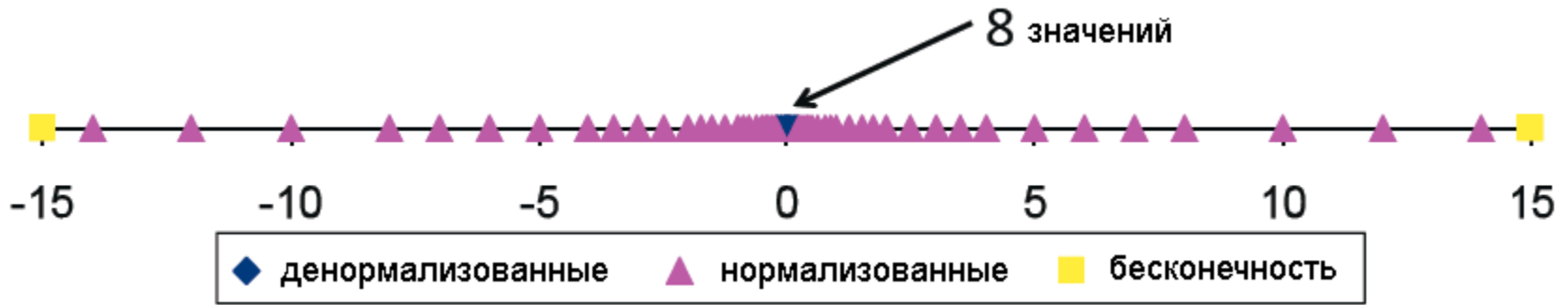


1      4 бита

3 бита

	s	exp	frac	E	Value	
Ненормализованные числа	0	0000	000	-6	0	
	0	0000	001	-6	$1/8 * 1/64 = 1/512$	Близкие к 0
	0	0000	010	-6	$2/8 * 1/64 = 2/512$	
	...					
	0	0000	110	-6	$6/8 * 1/64 = 6/512$	
	0	0000	111	-6	$7/8 * 1/64 = 7/512$	Наибольшее ненормализованное
Нормализованные числа	0	0001	000	-6	$8/8 * 1/64 = 8/512$	Наименьшее нормализованное
	0	0001	001	-6	$9/8 * 1/64 = 9/512$	
	...					
	0	0110	110	-1	$14/8 * 1/2 = 14/16$	
	0	0110	111	-1	$15/8 * 1/2 = 15/16$	Ближайшее к 1 снизу
	0	0111	000	0	$8/8 * 1 = 1$	
	0	0111	001	0	$9/8 * 1 = 9/8$	Ближайшее к 1 сверху
	0	0111	010	0	$10/8 * 1 = 10/8$	
	...					
	0	1110	110	7	$14/8 * 128 = 224$	
0	1110	111	7	$15/8 * 128 = 240$	Наибольшее нормализованное	
	0	1111	000	n/a	inf	

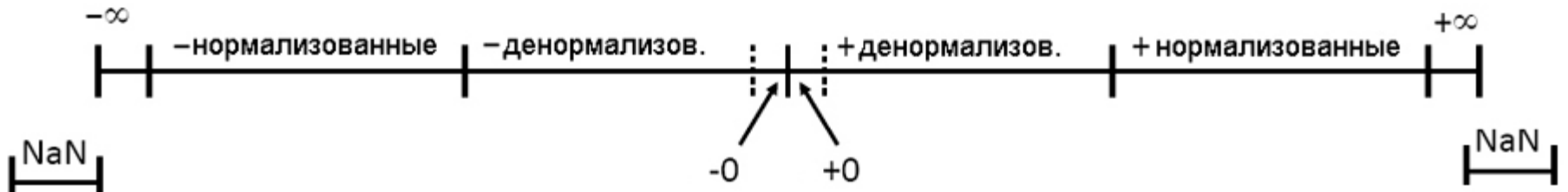
# 8-разрядные числа с плавающей точкой



Центральная область более крупно

# Важные частные случаи

	exp	frac	Численное значение
◆ Нуль	00...00	00...00	0.0
◆ Наим. положит. денорм.	00...00	00...01	
◆ float $\approx 1.4 \times 10^{-45}$			$2^{-23} \times 2^{-126}$
◆ double $\approx 4.9 \times 10^{-324}$			$2^{-52} \times 2^{-1022}$
◆ Наиб. положит. денорм.	00...00	11...11	
◆ float $\approx 1.18 \times 10^{-38}$			$(1.0 - \epsilon) \times 2^{-126}$
◆ double $\approx 2.2 \times 10^{-308}$			$(1.0 - \epsilon) \times 2^{-1022}$
◆ Наим. положит. норм.	00...01	00...00	
◆ float			$1.0 \times 2^{-126}$
◆ double			$1.0 \times 2^{-1022}$
◆ Единица	01...11	00...00	1.0
◆ Наиб. положит. норм.			
◆ float $\approx 3.4 \times 10^{38}$			$(2.0 - \epsilon) \times 2^{127}$
◆ double $\approx 1.8 \times 10^{308}$			$(2.0 - \epsilon) \times 2^{1023}$



# Операции над числами с плавающей точкой

◇ По определению:

$$x +_{FP} y = Round(x + y)$$

$$x \times_{FP} y = Round(x \times y)$$

где  $Round()$  означает округление

◇ Выполнение операции

- ◇ Сначала вычисляется точный результат (получается более длинная мантисса, чем запоминаемая, иногда в два раза)
- ◇ Потом фиксируется исключение (например, переполнение)
- ◇ Потом результат округляется, чтобы поместиться в поле *frac*



## Умножение чисел с плавающей точкой

◇  $(-1)^{s1} \cdot M1 \cdot 2^{e1} \times (-1)^{s2} \cdot M2 \cdot 2^{e2}$

◇ Точный результат  $(-1)^s \cdot M \cdot 2^e$

- ◆ Знак  $s$   $s1 \wedge s2$
- ◆ Значащие цифры  $M$   $M1 \times M2$
- ◆ Порядок  $e$   $e1 + e2$

◇ Преобразование

- ◆ Если  $M \geq 2$ , сдвиг  $M$  вправо с одновременным увеличением  $e$
- ◆ Если  $e$  не помещается в поле  $exp$ , переполнение
- ◆ Округление  $M$ , чтобы оно поместилось в поле  $frac$

◇ Основные затраты на перемножение мантисс

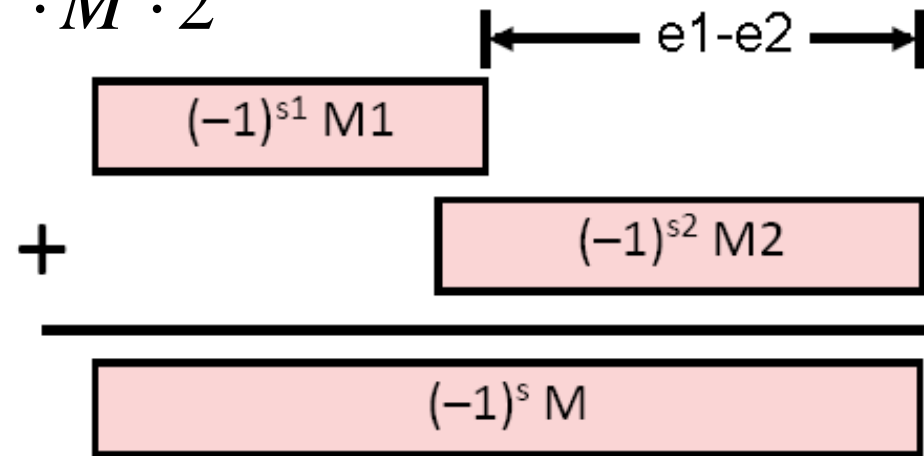
# Сложение чисел с плавающей точкой

$$\diamond \quad (-1)^{s1} \cdot M1 \cdot 2^{e1} + (-1)^{s2} \cdot M2 \cdot 2^{e2}$$

Пусть  $e1 > e2$

$$\diamond \quad \text{Точный результат } (-1)^s \cdot M \cdot 2^e$$

- ◆ Знак  $s$  и значащие цифры  $M$  вычисляются как показано на рисунке



- ◆ Порядок суммы –  $e1$

## ◆ Преобразование

- ◆ Если  $M \geq 2$ , сдвиг  $M$  вправо с одновременным увеличением  $e$
- ◆ Если  $M < 1$ , сдвиг  $M$  влево на  $k$  позиций с одновременным вычитанием  $k$  из  $e$
- ◆ Если  $e$  не помещается в поле  $exp$ , переполнение
- ◆ Округление  $M$ , чтобы оно поместилось в поле  $frac$

# Плавающие типы языка Си

## float, double, long double

- ◆ **Операции над данными с плавающей точкой.**
  - ◆ *Одноместные*: изменение знака («одноместный минус»:  $-$ ), одноместный плюс ( $+$ ).
  - ◆ *Двухместные*: сложение ( $+$ ), вычитание ( $-$ ), умножение ( $*$ ), деление ( $/$ ).
- ◆ **Порядок выполнения арифметических операций в выражениях (приоритет).**
  - ◆ самый низкий приоритет у двуместных  $+$  и  $-$ ,
  - ◆ более высокий приоритет у двуместных  $*$  и  $/$ ,
  - ◆ еще более высокий приоритет у одноместных  $+$  и  $-$ .
  - ◆ В выражениях без скобок операции с более высоким приоритетом выполняются раньше.
  - ◆ Скобки позволяют изменить порядок выполнения операций.

## Пример 1. Вычисление суммы 5 чисел типа `float`

(мантисса – 6 десятичных цифр, порядок – 2 десятичных цифры):

$$0.231876 * 10^{02} + 0.645391 * 10^{-03} + 0.231834 * 10^{-01} + 0.245383 * 10^{-02} + 0.945722 * 10^{-03} =$$

$$\text{a) } \mathbf{0.231876 * 10^{02} + 0.645391 * 10^{-03} + 0.231834 * 10^{-01} + 0.245383 * 10^{-02} + 0.945722 * 10^{-03} = 0.2321\mathbf{\underline{47}} * 10^{02};}$$

$$23.1876 + 0.000645391 = 23.188245391 = 23.1882 = 0.231882 * 10^{02};$$

$$23.1882 + 0.0231834 = 23.2113834 = 23.2114 = 0.232114 * 10^{02};$$

$$23.2114 + 0.00245383 = 23.21385383 = 23.2138 * 10^{02};$$

$$23.2138 + 0.000945722 = 23.214745722 = 23.2147 = 0.232147 * 10^{02};$$

$$\text{b) } \mathbf{0.645391 * 10^{-03} + 0.9457 * 10^{-03} + 0.245383 * 10^{-02} + 0.231834 * 10^{-01} + 0.231876 * 10^{02} = 0.2321\mathbf{\underline{57}} * 10^{02};}$$

$$0.000645391 + 0.000945722 = 0.001591113 = 0.00159111 = 0.159111 * 10^{-02};$$

$$0.00159111 + 0.00245383 = 0.00494493 = 0.494493 * 10^{-02};$$

$$0.00494493 + 0.0231834 = 0.02812833 = 0.0281283 = 0.281283 * 10^{-01};$$

$$0.0281283 + 23.1876 = 23.2157283 = 23.2157 = 0.232157 * 10^{02};$$

## Пример 2. Вычисление разности плавающих чисел

(мантисса – 6 десятичных цифр, порядок – 2 десятичных цифры):

$$\mathbf{0.238617*10^{02} - 0.238616*10^{02} + 0.645391*10^{04} - 0.645392*10^{04} + 0.845791*10^{00} - 0.835790*10^{00} =}$$

a)

$$0.238617*10^{02} - 0.238616*10^{02} + 0.645391*10^{04} - 0.645392*10^{04} + 0.845791*10^{00} - 0.835790*10^{00} = \mathbf{0.100000*10^{-05}}$$
$$0.238617*10^{02} - 0.238616*10^{02} = 23.8617 - 23.8616 = 0.0001 = \mathbf{0.100000*10^{03}}$$
$$0.100000*10^{-03} + 0.645391*10^{04} = 0.0001 + 6453.91 = 6453.9101 = \mathbf{0.645391*10^{04}}$$
$$0.645391*10^{04} - 0.645392*10^{04} = -0.000001*10^{04} = -\mathbf{0.100000*10^{-01}}$$
$$-0.100000*10^{-01} + 0.845791*10^{00} = -0.01 + 0.845791 = 0.835791 *10^{00}$$
$$0.835791 *10^{00} - 0.835790*10^{00} = \mathbf{0.000001*10^{00} = 0.100000*10^{-05}}$$

b)

$$0.238617*10^{02} + 0.645391*10^{04} + 0.845791*10^{00} - (0.238616*10^{02} + 0.645392*10^{04} + 0.835790*10^{00}) = \mathbf{0.100000*10^{00}}$$
$$0.238617*10^{02} + 0.645391*10^{04} = 23.8617 - 6453.91 = 6478.6 = \mathbf{0.647777*10^{04}}$$
$$0.647777*10^{04} + 0.845791*10^{00} = 6477.77 + 0.845791 = 6478.615791 = \mathbf{0.647862*10^{04}}$$
$$0.238616*10^{02} + 0.645392*10^{04} = 23.8616 + 6453.92 = 6477.7816 = \mathbf{6477.78*10^{04}}$$
$$6477.78*10^{04} + 0.835790*10^{00} = 6477.78 + 0.835790 = 6478.61579 = \mathbf{0.647852*10^{04}}$$
$$0.647862*10^{04} - 0.647852*10^{04} = \mathbf{0.000010*10^{04} = 0.100000*10^{-00}}$$

## **Выводы**

- (1) **При вычислении суммы чисел с одинаковыми знаками** необходимо упорядочить слагаемые по возрастанию и складывать, начиная с наименьших слагаемых.
- (2) **При вычислении суммы чисел с разными знаками** необходимо сначала сложить все положительные числа, потом – все отрицательные числа и в конце выполнить одно вычитание.
- (3) **Вычитание** (сложение чисел с противоположными знаками) **часто приводит к потере точности**, которая у чисел с плавающей точкой определяется количеством значащих цифр в мантиссе (при вычитании двух близких чисел мантисса «исчезает», что ведет к резкой потере точности).  
**Итак, чем меньше вычитаний, тем точнее результат.**

Значащими цифрами числа с плавающей точкой называются все цифры его мантиссы за исключением нулей, стоящих в ее конце. Например, у числа  $0.67000890000 * 10^3$  все цифры, выделенные жирным шрифтом, значащие. При вычитании двух близких чисел почти все значащие цифры пропадают. Например,  $0.67000890 * 10^3 - 0.67000880 * 10^3 = 0.00000010 * 10^3 = 0.10 * 10^{-4}$ . Таким образом, у результата всего одна значащая цифра, хотя у операндов было по 7 значащих цифр.