Курс «Алгоритмы и алгоритмические языки»
1 семестр 2018/2019

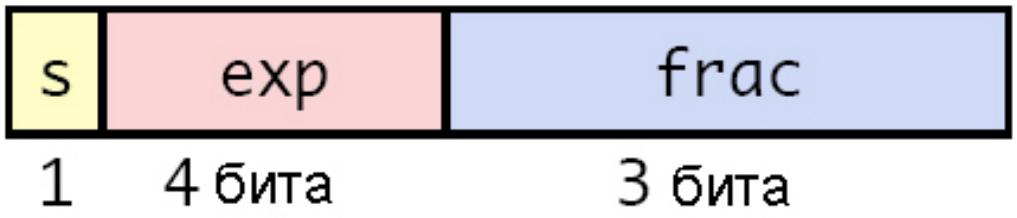
Лекция 16

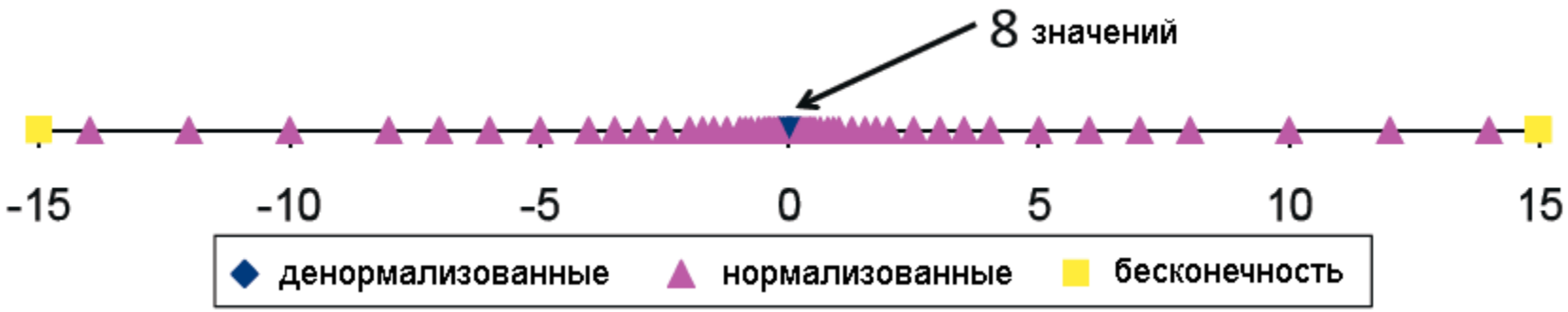
8-разрядные числа с плавающей точкой (положительные)

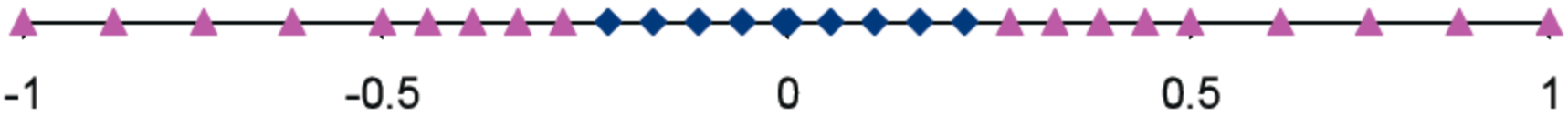
1 4 бита 3 бита

	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



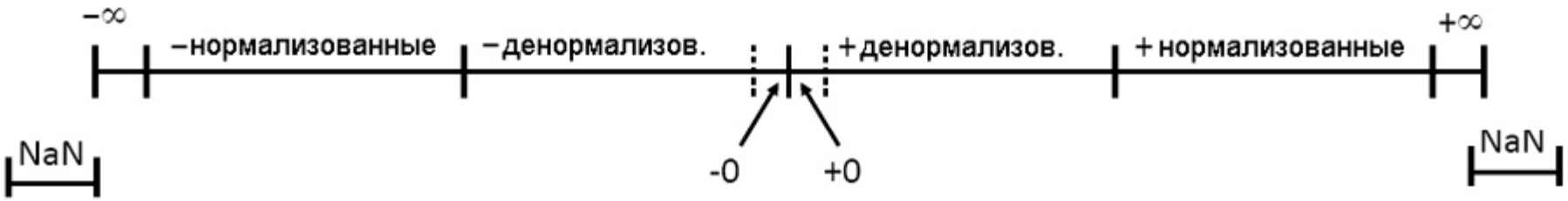




Центральная область более крупно

Важные частные случаи

	exp	frac	Численное значение
◈ Нуль	00…00	00…00	0.0
◈ Наим. положит. денорм.	00…00	00…01	
♦ float ≈ 1.4×10^{-45}			$2^{-23}\times2^{-126}$
♦ double ≈ 4.9×10^{-324}			$2^{-52}\times2^{-1022}$
◈ Наиб. положит. денорм.	00…00	11…11	
♦ float ≈ 1.18×10^{-38}			$(1.0 - \varepsilon)\times2^{-126}$
♦ double ≈ 2.2×10^{-308}			$(1.0 - \varepsilon)\times2^{-1022}$
◈ Наим. положит. норм.	00…01	00…00	
♦ float			1.0×2^{-126}
♦ double			1.0×2^{-1022}
◈ Единица	01…11	00…00	1.0
◈ Наиб. положит. норм.			
♦ float ≈ 3.4×10^{38}			$(2.0 - \varepsilon)\times2^{127}$
♦ double ≈ 1.8×10^{308}			$(2.0 - \varepsilon)\times2^{1023}$



Операции над числами с плавающей точкой

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

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

  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 $frac$

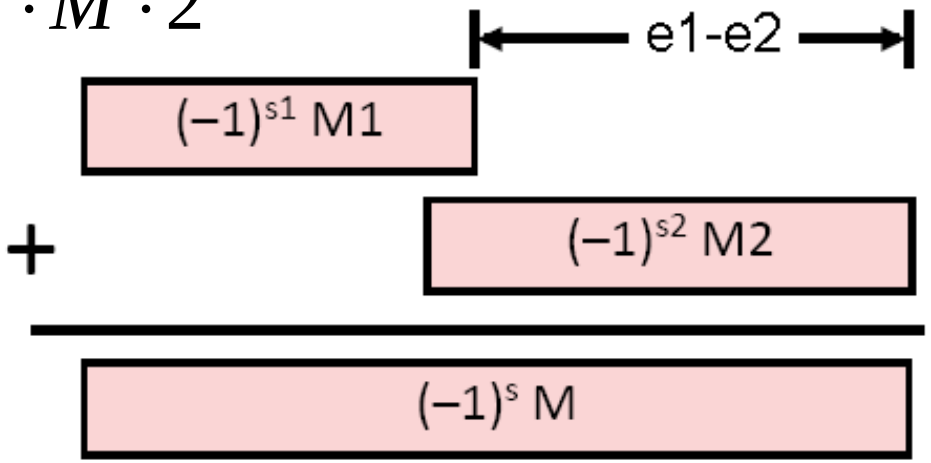
Умножение чисел с плавающей точкой

- $(-1)^{s1} \cdot M_1 \cdot 2^{e1} \times (-1)^{s2} \cdot M_2 \cdot 2^{e2}$
- Точный результат $(-1)^{s} \cdot M \cdot 2^{e}$
  - Знак s $s1 \wedge s2$
  - Значащие цифры M $M_1 \times M_2$
  - 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

- $(-1)^{s1} \cdot M_1 \cdot 2^{e1} + (-1)^{s2} \cdot M_2 \cdot 2^{e2}$
  Пусть $e1 > e2$

- 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 $frac$

Пример 1. Вычисление суммы 5 чисел типа **`float`**
(мантисса – 6 десятичных цифр, порядок – 2 десятичных цифры):

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

a) $\mathbf{0.231876*10^{02} + 0.645391*10^{-03} + 0.231834*10^{-01} + 0.245383*10^{-02} + 0.945722*10^{-03} = 0.2321\underline{4}7*10^{02}}$;

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};

b) $\mathbf{0.645391*10^{-03} + 0.9457*10^{-03} + 0.245383*10^{-02} + 0.231834*10^{-01} + 0.231876*10^{02} = 0.2321\underline{5}7*10^{02}}$;

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):

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

a) $0.238617*10^{02} – 0.238616*10^{02} + 0.645391*10^{04} – 0.645392*10^{04} + 0.845791*10^{00} – 0.835790*10^{00} =$ **$0.100000*10^{-05}$**

$0.238617*10^{02} – 0.238616*10^{02} = 23.8617 – 23.8616 = 0.0001 =$ **$0.100000*10^{03}$**

$0.100000*10^{-03} + 0.645391*10^{04} = 0.0001 + 6453.91 = 6453.9101 =$ **$0.645391*10^{04}$**

$0.645391*10^{04} – 0.645392*10^{04} = – 0.000001*10^{04} =$ **$– 0.100000*10^{-01}$**

$– 0.100000*10^{-01} + 0.845791*10^{00} = – 0.01 + 0.845791 = 0.835791 *10^{00}$

$0.835791 *10^{00} – 0.835790*10^{00} =$ **$0.000001*10^{00} = 0.100000*\underline{10^{-05}}$**

b) $0.238617*10^{02} + 0.645391*10^{04} + 0.845791*10^{00} – (0.238616*10^{02} + 0.645392*10^{04} + 0.835790*10^{00}) =$ **$0.100000*10^{00}$**

$0.238617*10^{02} + 0.645391*10^{04} = 23.8617 – 6453.91 = 6478.6 =$ **$0.647777*10^{04}$**

$0.647777*10^{04} + 0.845791*10^{00} = 6477.77 + 0.845791 = 6478.615791 =$ ***$0.647862*10^{04}$***

$0.238616*10^{02} + 0.645392*10^{04} = 23.8616 + 6453.92 = 6477.7816 =$ **$6477.78*10^{04}$**

$6477.78*10^{04} + 0.835790*10^{00} = 6477.78 + 0.835790 = 6478.61579 =$ ***$0.647852*10^{04}$***

$0.647862*10^{04} – 0.647852*10^{04}=$ **$0.000010*10^{04} = 0.100000*\underline{10^{-00}}$**

Выводы по операциям

(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.**6700089**0000 * 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.

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 **–**.

Режимы gcc для работы с плавающей точкой*

- **`https://gcc.gnu.org/wiki/FloatingPointMath`**
  Детальное резюме того, что бывает в **`gcc`**, и таблица преобразований, влияющих на результат вычислений
- **`-ffast-math`**: считать максимально быстро, но, возможно, нарушать стандарт IEEE-754
  - Полезно для тестирования, но не распространения финальной версии программы
- **`-fno-math-errno`**: не устанавливать переменную **`errno`** как результат ошибочного выполнения математических функций
  - Можно обойтись и без этого, но зависит от библиотеки Си
  - Компилятор может заменять вызовы функций инструкциями процессора (например, sqrt)
- **`-fno-trapping-math`**: считать, что вычисления с плавающей точкой не могут вызывать исключений процессора (traps)
  - Т.е. вы гарантируете отсутствие в своем коде ситуаций, вызывающих деления на ноль, переполнения, некорректные операции
  - Компилятор может более свободно комбинировать, переставлять, удалять операции с плавающей точкой
- David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.* 23, 1 (March 1991), 5-48
  https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Сложность алгоритмов

- ***Размер входа*:** числовая величина, характеризующая количество входных данных (например – длина битовой записи чисел - параметров алгоритма)
- ***Сложность в наихудшем случае*:** функция размера входа, отражающая максимум затрат на выполнение алгоритма для данного размера
  - временнАя сложность
  - пространственная сложность (затраты памяти)
  - часто оценивают не все затраты, а только самые “дорогие” операции
- ***Сложность в среднем*:** функция размера входа, отражающая средние затраты на выполнение алгоритма для входа данного размера (учет вероятностей входа)
- ***Асимптотические оценки сложности*:** O-нотация (оценка сверху), точная O-оценка, Θ-оценка.
- Подробности: С.А. Абрамов. Лекции о сложности алгоритмов. М.: МЦНМО, 2009

Формальная постановка задачи поиска по образцу

- Даны *текст* – массив $T[N]$ длины N и *образец* – массив $P[m]$ длины $m \le N$, где значениями элементов массивов T и P являются символы некоторого *алфавита* A.
- Говорят, что образец P входит в текст T со сдвигом s, если $0 \le s \le N - m$ и для всех $i = 1, 2, \ldots, m$ $T[s + i] = P[i]$.
- Сдвиг $s(T, P)$ называется *допустимым*, если P входит в T со сдвигом $s = s(T, P)$, и *недопустимым* в противном случае.
- ***Задача поиска подстрок*** состоит в нахождении множества допустимых сдвигов $s(T, P)$ для заданного текста T и образца P.

Формальная постановка задачи поиска по образцу

- ***Терминология***. Пусть строки *x, y, w* $\in A^*$, $\varepsilon \in A^*$ - пустая строка; |*x*| - длина строки *x*;

  xy – конкатенация строк *x* и *y*; |*xy*| = |*x*| + |*y*|;

  x = *wy* -> *w* – *префикс* (начало) *x* (обозначение $w \prec x$);

  x = *yw* -> *w* – *суффикс* (конец) *x* (обозначение $w \succ x$);

  если *w* – префикс или суффикс *x*, то |*w*| $\leq$ |*x*|;

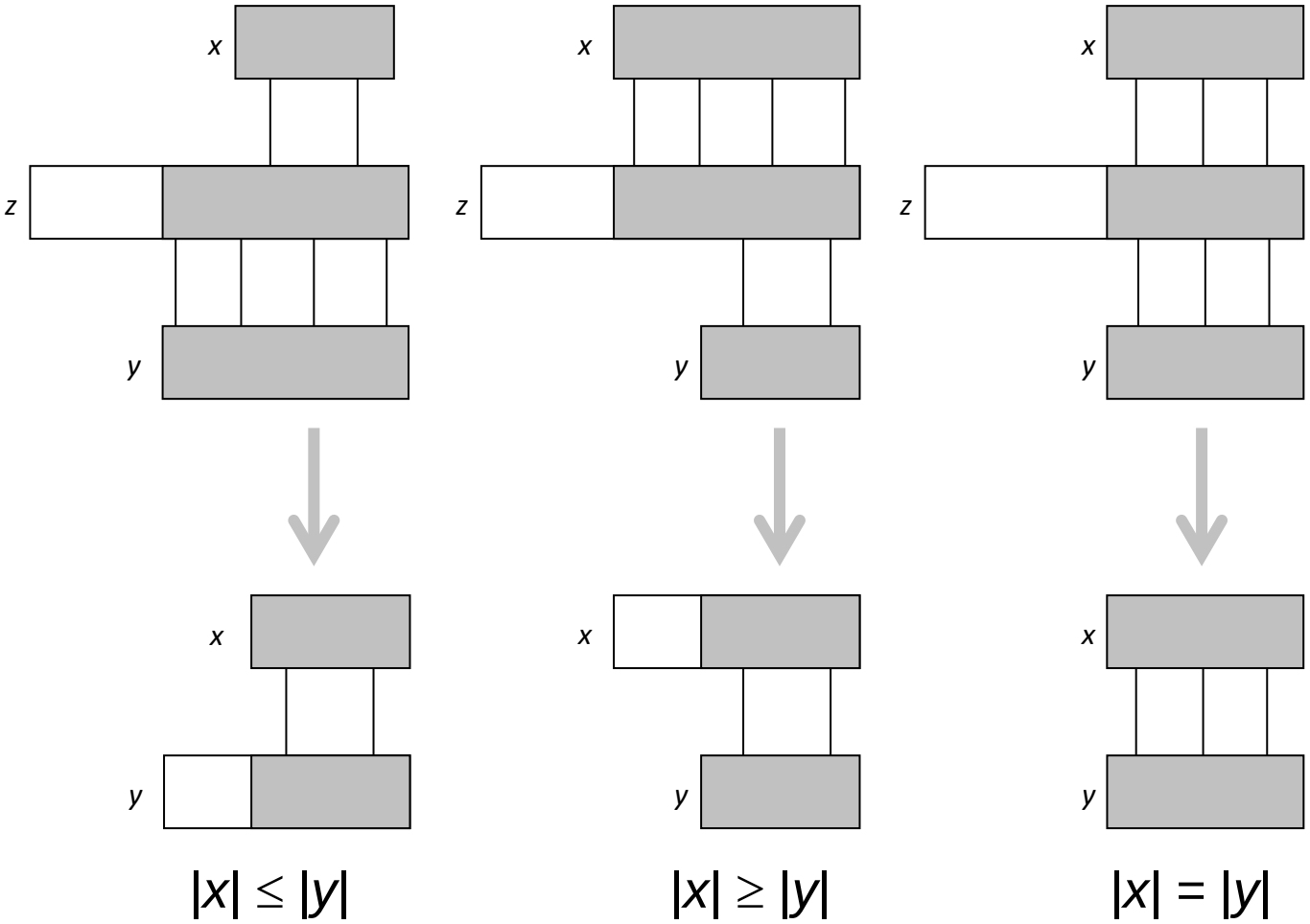
  отношения префикса и суффикса *транзитивны*.

  Для любых *x, y* $\in A^*$ и любого *a* $\in A$ соотношения $x \succ y$ и $xa \succ ya$ *равносильны*.

- Если *S* = S[*r*] – строка длины *r*, то ее префикс длины *k*, *k* $\leq$ *r* будет обозначаться S_k = *S*[*k*]; ясно, что $S_0 = \varepsilon$, S_r = S.

Лемма (*о двух суффиксах*)

- Пусть *x*, *y* и *z* – строки, для которых $x \succ z$ и $y \succ z$.
  Тогда если |*x*| ≤ |*y*|, то $x \succ y$, если |*x*| ≥ |*y*|, то $y \succ x$,
  если |*x*| = |*y*|, то $x = y$.



Простой алгоритм

- Проверка совмещения двух строк: посимвольное сравнение слева направо, которое прекращается (с отрицательным результатом) при первом же расхождении.
- Оценка скорости сравнения строк x и y – $\Theta(t + 1)$, где t – длина наибольшего общего префикса строк x и y.

```
for (s = 0; s <= n – m; s++) {
  for (i = 0; i < m && P[i] == T[s + i]; i++)
    ;
  if (i == m)
    printf ("%d\n", s);
}
```

- Время работы в худшем случае $\Theta((n – m + 1)\cdot m) \sim \Theta(nm)$.

  Причина: информация о тексте T, полученная при проверке данного сдвига s, никак не используется при проверке следующих сдвигов. Например, если для образца **dddc** сдвиг $s = 0$ допустим, то сдвиги $s = 1, 2, 3$, недопустимы, так как **T[3] == c**.