Словарь языка J

Roger K.W. Hui
Kenneth E. Iverson

русский перевод: Константин Л. Метлов.

Версия от: 2011-5-3
www.jsoftware.com

Содержание Словаря Титульный лист

Введение

J -- язык программирования общего назначения, реализованный для множества различных вычислительных машин. Несмотря на простую структуру и доступность для тех, кто знаком с математическими понятиями и записью, некоторые особенности делают **J** сложным для изучения теми, кто привык к использованию более традиционных языков программирования.

Чтобы повысить доступность этого введения для опытных программистов, в нем постоянно подчеркиваются особенности, отличающие **J** от других языков. К ним относятся:

1. Мнемоническое одно- или двух- буквенное обозначение примитивов.

2. Отсутствие приоритета операций.

3. Систематическое использование *двойственности* функций, которые, как знак "минус" в арифметике, могут означать одну функцию, когда вызваны с двумя аргументами (*вычитание* в случае **-**), и другую, когда вызваны с одним (*перемена знака* в случае `-`).

4. Использование понятий из грамматики естественного языка, более подходящих для описания грамматики **J**, чем термины, используемые обычно в математике и других языках программирования. Тоесть функция (такая как сложение) называется также *глаголом* (поскольку она производит действие), а нечто, изменяющее действие глагола (понятие, отсутствующее во многих языках программирования), называется, соответственно, *наречием*.

5. Систематическое использование наречий и союзов для изменения действия глаголов позволяет представить богатый набор операций, используя сравнительно небольшое множество глаголов. Например, `+/a` обозначает сумму по элементам `a` , `*/a` обозначает произведение по `a` , `a */ b` есть таблица умножения `a` и `b` .

6. Работа с векторами, матрицами и другими массивами как с примитивными обьектами.

7. Использование *функционального* или *неявного* программирования, не требующего явного упоминания

аргументов определяемой функции (программы); использование присваивания (имен) для функций (как в `sum=:+/` и `mean=:sum % #`).

Следующие разделы содержат комментированные протоколы сессий **J**. Комментарии следует читать только после изучения соответствующей сессии (и, возможно, самостоятельных экспериментов с ее вариациями на компьютере). Эти разделы нужно изучать, имея под рукой компьютер и словарь **J**, не забывая выполнять упражнения. Нетерпеливый читатель может сразу перейти к Примерам.

1. Мнемоника

Далее показан комментированный протокол компьютерной сессии. Результат каждого предложения дается с отступом влево. Сначала, не глядя на комментарии, попробуйте пересказать на русском значение каждого примитива слева, чтобы понять отношения между похожими символами. Например, `<` читается как "*меньше*", а `<.` как "*меньшее* (тоесть минимум)". Потом посмотрите на комментарии и сравните со своими собственными.

```
   7<5                          Меньше
0                               Ноль обозначает ложь.

   7<.5                         Меньшее
5

   7>5                          Больше
1                               Единица означает истину (a la George Boole)

   7>.5                         Большее
7

   10^3                         В Степени (a la Augustus De Morgan)
1000

   10^.1000                     Логарифм
3

   7=5                          Равно
0

   b=: 5                        Есть (присваивание или привязка)
   7<.b
5

   Min=: <.                     Min есть <.
   power=: ^                    power есть ^
   gt=: >                       gt есть >

   10 power (5 Min 3)
1000
```

Упражнения

1.1 Введите в компьютер следующие предложения и посмотрите на результат. Попытайтесь дать подходящие имена использованным примитивам (таким как `* , +.` и `*.`), отметьте особенности их поведения.

```
a=:0 1 2 3
b=:3 2 1 0
a+b
a*b
a-b
a%b
a^b
a^.b
a<b
a>b
(a<b)+(a>b)
(a<b)+.(a>b)
```

Сравните ваши заметки со следующим:

a) Отрицательное число "минус 3" обозначается `_3`. Символ подчеркивания `_` является такой же частью представления отрицательного числа, как десятичная точка является частью представления одной второй, записанной в виде `0.5`. Важно не путать знак "минус" `_` и символ, обозначающий вычитание (т.е. `-`).

b) Деление (`%`) на ноль дает бесконечность, обозначаемую отдельно стоящим подчеркиванием.

c) Логарифм 2 по основанию 1 равен бесконечности, а логарифм `0` по основанию 3 равен минус бесконечности (`__`).

d) Поскольку отношение `5<7` истинно, и результат `5<7` равен 1, можно сказать, что "истина" и "ложь" обозначаются обычными целыми 1 и `0`. Джордж Буль записывал эти значения так-же, обозначая, кроме того, символом `+` *булевскую* функцию *или*. Мы используем `+.` для *или*, во избежание путаницы с подобной (но все-же другой) функцией *сложения* (обозначаемой `+`).

1.2 По типу `Min=: <.` , придумайте, присвойте и используйте имена для каждого из встреченных здесь примитивов.

2. Двойственность

Не смотрите сразу на комментарии справа, а попробуйте дать свои.

```
   7-5
2
   -5
_5
```

Функция в предложении `7-5` применяется к дв
аргументам, и производит вычитание, но в
предложении `-5` она применяется к одному ар
производит перемену знака.

Заимствуя термин "валентность" из химии, мож
сказать, что символ **-** имеет *переменную* вален
определяемую из контекста.

```
   7%5
1.4
```

```
   %5
0.2
```

Эта двойственность `-` знакомая из арифметики
распространена здесь и на другие функции.

```
   3^2
9
```

```
   ^2
7.38906
```

Экспонента (тоесть `2.71828^2`)

```
   a=: i. 5
   a
0 1 2 3 4
```

Функция *целые* или *список целых*

список или *вектор*

```
   a i. 3 1
3 1
```

Функция *индекс* или *индекс в*

```
   b=: 'Canada'
   b i. 'da'
4 1
```

Одинарные кавычки обрамляют текстовую стро

```
   $ a
5
```

Функция *взять размерность*

```
   3 4 $ a
0 1 2 3
4 0 1 2
3 4 0 1
```

Функция *придать размерность*
Таблица или *матрица*

```
   3 4 $ b
Cana
daCa
nada
```

```
   %a
_ 1 0.5 0.333333 0.25
```

Функции применимы к спискам
Отдельно стоящий символ `_` означает *бесконеч*

Упражнения

2.1 Введите следующие предложения (и, возможно, подобные, но с другими аргументами), посмотрите на результаты и опишите действие обоих (*монадного* и *диадного*) вариантов каждой функции:

```
a=: 3 1 4 1 5 9
b=: 'Canada'
#a
1 0 1 0 1 3 # a
1 0 1 0 1 3 # b
/: a
/: b
a /: a
a /: b
b /: a
b /: b
c=: 'can''t'
c
#c
c /: c
```

2.2 Сделайте таблицу функций, встреченных до сих пор. Сравните ее со следующей таблицей (где звездочкой разделяются монадный и диадный случаи, как в "Поменять Знак • Вычесть"):

+	• Прибавить	• Или	
-	Поменять Знак • Вычесть		
*	• Умножить	• И	
%	Обратить • Разделить		
^	Экспонента • В Степени	• Логарифм	
<	• Меньше	• Меньшее	
>	• Больше	• Большее	
=	• Равно		Есть (присваивание)
i		Целые • Индекс В	
$	Размерность • Придать Размерность		

/ Упоряд. Перестановка • Упорядочить

Количество Элементов • Копировать

2.3 Попробуйте заполнить пустые места в таблице Упражнения 2.2, экспериментируя на компьютере с соответствующими выражениями. Введите, например, `^.10` и `^. 2.71828` , чтобы исследовать отсутствующий (монадный) вариант `^.` . Введите `%: 4` , `%: -4` и `+%: -4` для исследования случая `%` с двоеточием.

Не тратьте слишком много времени на результаты, которые не понятны сразу (такие как, возможно, комплексные числа или *упаковки*, производимые монадой `<`). К ним лучше вернуться после того как мы пройдем несколько следующих разделов. Заметьте, что эффект некоторых функций становится заметным только когда они применяются к аргументам, отличным от положительных целых: попробуйте `<.1 2 3 4` и `<.3.4 5.2 3.6` для определения эффекта монады `<.` .

2.4 Если `b=: 3.4 5.2 3.6` , то `<.b` возвращает аргумент `b` , округленный *вниз* до ближайшего целого. Напишите и протестируйте предложение, округляющее аргумент `b` до *ближайшего* целого.

Ответ: `<.(b+0.5)` или `<.b+0.5` или `<.b+1r2`

2.5 Введите `2 4 3 $ i. 5` , чтобы увидеть пример *массива ранга* `3` (количество измерений равно 3) или *отчета* (для двух лет, четырех кварталов по три месяца в каждом).

2.6 Введите несколько раз подряд `?9` и определите действие функции `?` . Потом введите `t=: ?3 5 $ 9` , чтобы получить таблицу для дальнейших экспериментов.

Ответ: `?` -- генератор (псевдо-) случайных чисел; `?n` выдает случайный элемент из списка `i.n`

3. Глаголы и Наречия

В предложении `%a` из Раздела 2, `%` "действует на" `a`, производя некоторый результат. В этом смысле, `%a` аналогично *глаголу*, действующему на существительное. В дальнейшем, мы будем использовать термин *глагол* вместе (или наравне с) математическим термином *функция*, использованным до сих пор.

Предложение `+/ 1 2 3 4` эквивалентно `1+2+3+4`; Здесь, наречие *между* (`/`), в применении к *своему* глагольному аргументу `+` , производит новый глагол, вставляющий глагол `+` между элементами своего аргумента `1 2 3 4`. Если аргумент наречия `/` другой, он обрабатывается подобным образом:

```
   */b=:2 7 1 8 2 8
1792

   <./b
1

   >./b
8
```

Глагол, получающийся применением наречия, может (как и примитивный глагол) иметь монадный и диадный варианты использования. В силу этой двойственности, наречие `/` называется либо *между* либо *таблично*. Диадный вариант `/` производит *таблицу*. Например:

```
   2 3 5 +/ 0 1 2 3
2 3 4 5
3 4 5 6
5 6 7 8
```

Глаголы `over=:({.;}.)@":@,` и `by=:' '&;@,.@[,.]` полезны как утилиты (для использования, а не для немедленного изучения), они могут помочь в интерпретации *таблиц функций*, таких как таблица сложения, показанная ниже. Например (предварительно введя определения `over` и `by`):

```
   a=: 2 3 5
   b=: 0 1 2 3

   a by b over a +/ b
+-+-------+
| |0 1 2 3|
+-+-------+
|2|2 3 4 5|
|3|3 4 5 6|
|5|5 6 7 8|
```

```
+-+-------+
   b by b over b </ b
+-+-------+
| |0 1 2 3|
+-+-------+
|0|0 1 1 1|
|1|0 0 1 1|
|2|0 0 0 1|
|3|0 0 0 0|
+-+-------+
```

Упражнения

3.1 Введите `d=: i.5` , а потом предложения `st=: d-/d` и `pt=: d^/d` для построения таблиц вычитания и степени.

3.2 Постройте таблицы других функций из предыдущих разделов, включая отношения `<` , `=` и `>` а также *меньшее* и *большее*.

3.3 Примените глаголы `|.` и `|:` к различным таблицам, и попробуйте описать их действие.

3.4 Функция *транспонировать* `|:` изменяет таблицу вычитания, но не производит никакого эффекта над таблицей умножения. Назовите свойство функций, таблицы которых не меняются при транспонировании.

Ответ: они коммутативны

3.5 Введите `d by d over d!/d` и дайте определение диады `!` .

Ответ: `!` -- это биномиальный коэффициент или функция *количества выборок*; `3!5` есть количество различных способов, которыми можно выбрать три вещи из пяти.

4. Пунктуация

Для *пунктуации* в русском используются несколько специальных символов, определяющих порядок интерпретации фразы. Например:

Казнить, нельзя помиловать.
Казнить нельзя, помиловать.

Математика тоже использует различные конструкции (в первую очередь скобки) для указания порядка интерпретации, или, как это часто называется, *порядка выполнения*. Кроме того, в математике используется набор правил для интерпретации выражений без скобок, включающий установку относительных приоритетов функций. Например, *степень* выполняется перед *умножением*, которое, в свою очередь, выполняется перед *сложением*.

В **J** для пунктуации используются скобки, и следующие правила при их отсутствии:

Правый аргумент глагола -- значение всей фразы, стоящей справа от него.
Сначала применяются наречия. Тоесть, выражение `a */ b` означает `a (*/) b`, но не `a(*/b)`.

Например:

```
   a=:5
   b=:3
   (a*a)+(b*b)
34

   a*a+b*b
70

   a*(a+(b*b))
70

   (a+b)*(a-b)
16

   a (+*-) b
16
```

Последнее предложение включает *изолированную* цепочку из трех глаголов `+*-` , интерпретацию которой мы пока не обсуждали. Такая цепочка называется *вилкой*. Два последних предложения эквивалентны.

Вилка имеет и монадную интерпретацию, как иллюстрировано для

среднего ниже:

```
   c=:2 3 4 5 6
   (+/ % #) c
4

   (+/c)%(#c)
4
```

Глагол # дает количество элементов в своем а

Упражнения

4.1 В математике, выражение $3x^4+4x^3+5x^2$ называется *многочленом*. Введите:

```
   x=: 2
   (3*x^4)+(4*x^3)+(5*x^2)
```

для вычисления многочлена при x равном 2.

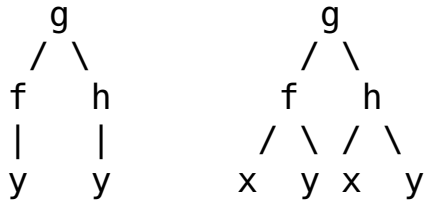
4.2 Заметьте, что, при стандартной математической иерархии приоритетов функций, для записи многочлена скобки не нужны. Запишите эквивалентное выражение без скобок.

Ответ: `+/3 4 5 * x ^ 4 3 2`

или (присваивая имена *коэффициентам* 3 4 5 и *степеням* 4 3 2), как: `+/c*x^e`

5. Вилки

Как мы уже видели, изолированная последовательность трех глаголов называется *вилкой*; ее монадный и диадный случаи определены как:



Приведенные выше диаграммы дают визуальную иллюстрацию вилки.

Не читая комментариев справа (используя тот факт, что `%:` обозначает взятие *корня*, а `]` *тождественную* функцию), попробуйте выразить на русском значение каждого из следующих предложений:

```
   a=: 8 7 6 5 4 3
   b=: 4 5 6 7 8 9
   2 %: b                    Квадратный корень b
2 2.23607 2.44949 2.64575 2.82843 3

   3 %: b                    Кубический корень b
1.5874 1.70998 1.81712 1.91293 2 2.08008

   (+/ % #) b                Среднее арифметическое, или просто "среднее
6.5

   (# %: */) b               Среднее геометрическое
6.26521

   (] - (+/ % #)) b          Центрировать вокруг среднего (две вилки)
_2.5 _1.5 _0.5 0.5 1.5 2.5

   (] - +/ % #) b            Две вилки (меньше скобок)
_2.5 _1.5 _0.5 0.5 1.5 2.5


   a (+ * -) b               Диадный случай вилки
48 24 0 _24 _48 _72

   (a^2)-(b^2)
48 24 0 _24 _48 _72

   a (< +. =) b              Меньше или равно
0 0 1 1 1 1
```

```
   a<b
0 0 0 1 1 1

   a=b
0 0 1 0 0 0

   a (<: = < +. =) b
1 1 1 1 1 1

   2 ([: ^ -) 0 1 2
7.38906 2.71828 1

   evens=: [: +: i.
   evens 7
0 2 4 6 8 10 12

   odds=: [: >: evens
   odds 7
1 3 5 7 9 11 13
```

Тавтология (<: есть *меньше либо равно*)

Шапка дает монадный случай

+: *удваивает*

>: *прибавляет 1*

Упражнения

5.1 Введите 5#3 и подобные выражения для исследования диады # . Затем опишите значение следующего предложения:

```
(# # >./) b=: 2 7 1 8 2
```

Ответ: #b повторений максимума в b

5.2 Закройте комментарии справа, запишите собственные интерпретации каждого предложения, сравните:

`(+/ % #) b`	Среднее по `b`
`(# # +/ % #) b`	`(n=:#b)` повторений среднего
`+/(##+/%#) b`	Сумма `n` средних
`(+/b)=+/(##+/%#) b`	Тавтология
`(*/b)= */(###%:*/) b`	Произведение элементов `b` есть произведен повторений среднего геометрического `b`

6. Программа

Выдаваемая перед концертом *программа*, описывает последовательность музыкальных произведений, которые будут там исполнены. Как видно из ее корней: *gram* и *pro*, программа -- нечто, написанное заранее о событиях, ею предписанных.

Подобным образом, вилка `+/ % #` из предыдущего раздела является программой, предписывающей вычисление среднего при применении ее к некоторому аргументу, как в выражении `(+/%#)b`. Однако, мы не будем обычно называть процедуру программой до тех пор, пока ей не присвоено имя, как показано ниже:

```
   mean=: +/ % #
   mean 2 3 4 5 6
4

   (geomean=: # %: */) 2 3 4 5 6
3.72792
```

Поскольку программа `mean` является новым *глаголом*, мы будем так же ссылаться на предложения типа `mean=: +/ % #` как на *определение глагола* (или просто *определение*), а на получающийся в их результате глагол как на *определенный* глагол или функцию.

Определенные глаголы могут учавствовать в определениях дополнительных глаголов способом, называемым иногда *структурированным программированием*. Например:

```
   MEAN=: sum % #
   sum=: +/
   MEAN 2 3 4 5 6
4
```

Ввод только глагола (без аргумента) показывает его определение, внешний союз (`!:`) позволяет установить вид этого отображения: коробочный, древовидный, линейный или скобочный. (Способ отображения глаголов можно так же настроить при помощи меню Edit|Configure...|View.) Таким образом:

```
   mean
+/ % #

   9!:3 (2 4 5)
   mean
+-----+-+-+
|+-+-+|%|#|
||+|/|| | |
|+-+-+| | |
```

```
+-----+-+-+
  +- / --- +
--+- %
  +- #
+/ % #
```

Упражнения

6.1 Введите `AT=: i. +/ i.` и исследуйте поведение программы `AT` при помощи выражний типа `AT 5` .

6.2 Определите и проверьте подобные таблицы функций для других диадных глаголов.

6.3 Определите программы:

```
tab=: +/
ft=: i. tab i.
test1=: ft = AT
```

Потом, применяя `test1` к различным целым аргументам, убедитесь, что `ft` эквивалентна `AT` из Упражнения 6.1. Введите `ft` и `AT` без аргументов, чтобы увидеть их определения.

6.4 Определите `aft=: ft f.` и, используя `test2=: aft = ft` , убедитесь в эквивалентности `ft` и `aft` . Отобразите их определения и опишите эффект наречия `f.` .

Ответ: Наречие `f.` *закрепляет* глагол, к которому оно применено, заменяя каждое использованное имя его значением.

6.5 Переопределите `tab` из Упражнения 6.3 введя `tab=: */` и посмотрите на эффект, производимый этим переопределением на `ft` и на его *фиксированный* вариант `aft` .

6.6 Определите `mean=: +/ % #` и опишите его поведение в применении к *таблице*, как в `mean t=: (i. !/ i.) 5` .

Ответ: Результат есть *среднее строк*, не *средние по строкам* табличного аргумента.

6.7 Запишите выражение для среднего *столбцов* `t` .

Ответ: `mean |: t` или `mean"1 t`

7. Союз "C"

Диада, например такая как ^ , может произвести семейство монадных функций:

```
   ]b=: i.7
0 1 2 3 4 5 6

   b^2                          Квадраты
0 1 4 9 16 25 36

   b^3                          Кубы
0 1 8 27 64 125 216

   b^0.5                        Квадратные корни
0 1 1.41421 1.73205 2 2.23607 2.44949
```

Союз *с* & позволяет *привязать* фиксированный аргумент к диаде, производя соответствующий монадный глагол. Например:

```
   square=: ^&2                 Квадрат (степень с 2)
   square b
0 1 4 9 16 25 36

   (sqrt=: ^&0.5) b             Квадратный корень
0 1 1.41421 1.73205 2 2.23607 2.44949
```

Левый аргумент можно привязать подобным образом:

```
   Log=: 10&^.                  Логарифм по основанию 10
   Log 2 4 6 8 10 100 1000
0.30103 0.60206 0.778151 0.90309 1 2 3
```

Глаголы с привязанными аргументами можно, конечно, использовать и в вилках. Например:

```
   in29=: 2&< *. <&9            Проверка на попадание в интервал
   in29 0 1 2 5 8 13 21
0 0 0 1 1 0 0

   IN29=: in29 # ]              Выборка попавших
   IN29 0 1 2 5 8 13 21
5 8

   LOE=: <+.=
   5 LOE 3 4 5 6 7
0 0 1 1 1

   integertest=: <. = ]         Монада <. дает пол (ближайшее меньшее цело
   integertest 0 0.5 1 1.5 2 2.5 3
1 0 1 0 1 0 1
```

```
   int=: integertest
   int (i.13)%3
1 0 0 1 0 0 1 0 0 1 0 0 1
```

Упражнения

7.1 В программе IN29 глагол # использован диадно. Введите фразы типа `(j=: 3 0 4 0 1) # i.5` для исследования # . Каким будет результат `#j#i.5` ? (попробуйте и `1j1#i.5`)

Ответ: `+/j`

7.2 Закройте ответы справа и примените следующие программы к различным спискам, чтобы выяснить (описав его на русском) предназначение каждой:

`test1=: >&10 *. <&100`	Проверить, если в интервале от 10 до 100
`int=: ] = <.`	Проверить, если целое
`test2=: int *. test1`	Проверить, если целое и от 10 до 100
`test3=: int +. test1`	Проверить, если целое или от 10 до 100
`sel=: test2 # ]`	Выбрать целые от 10 до 100

7.3 В предыдущем упражнении, закройте определения программ слева и попробуйте написать новые, производящие описанные эффекты.

7.4 Просмотрите еще раз использование *закрепляющего* наречия в Упражнениях 6.4-5, и поэкспериментируйте с его использованиям для программ из Упражнения 7.2.

8. Союз "Поверх"

Союз @ в применении к двум глаголам производит глагол, соответствующий применению первого *поверх* второго. Например:

```
   TriplePowersOf2=: (3&*)@(2&^)
   TriplePowersOf2 0 1 2 3 4
3 6 12 24 48

   CubeOfDiff=: (^&3)@-
   3 4 5 6 CubeOfDiff 6 5 4 3
_27 _1 1 27
```

```
   f=: ^@-
```

Правый глагол сначала применяется диадно, если возможно; левый применяется монадно.

```
   5 f 3
7.38906

   f 3
0.0497871

   g=: -@^
   5 g 3
_125

   g 3
_20.0855
```

Как и наречия, союзы выполняются перед глаголами. Их *левым* аргументом считается все предшествующее глагольное выражение. Соответственно, некоторые (но не все) скобки в предыдущих определениях можно опустить. Например:

```
   COD=: ^&3@-
   3 4 5 6 COD 6 5 4 3
_27 _1 1 27

   TPO2=: 3&*@(2&^)

   TPO2 0 1 2 3 4
3 6 12 24 48
```

```
   tpo2=: 3&*@2&^
|domain error
|   tpo2=:    3&*@2&^
```

Ошибка, поскольку союз @ определен только для глагольного правого аргумента

Упражнения

8.1 Закройте комментарии справа и опишите действие, производимое программами. Потом закройте программы и запишите их снова, используя описания на русском:

`mc=: (+/%#)@|:`	Среднее столбцов таблицы
`f=: +/@(^&2)`	Сумма квадратов элементов списка
`g=: %:@f`	Геометрическая длина списка
`h=:{&' *'@(</)`	Карта сравнения (диада)
`k=: i. h i.`	Карта (монада)
`map=: {&'+-* %#$'`	Карта из 7-ми символов
`MAP=: map@(6&<.)`	Расширенный диапазон для `map`
`add=: MAP@ (i.+/i.)`	Карта таблицы сложения

9. Разговорник

Запоминание списков слов -- сложный и неэффективный способ изучения нового языка, есть гораздо лучшие:

А) Беседа с человеком, в совершенстве знающим язык и стимулирующим Вас к разговору.

Б) Чтение материала, представляющего самостоятельный интерес.

В) Умение работать со словарем и знание грамматики дает независимость от учителя.

Г) Попытки *писать* на любую интересную тему.

Д) Внимание к *структуре* слов, превращающее известные слова в подсказку к значению неизвестных. Например, слово *программа* (уже проанализированное) имеет общее со словом *теле* (далеко) *грамма*, которое, в свою очередь, связано со словом *телефон*. Даже маленькие слова могут иметь структуру: *атом* означает неразрезаемый, от *а* (не) и *том* (как в словах *том* и *микротом*).

В случае **J**:

А) Компьютер служит для точной, неограниченной по времени и темам беседы.

Б) Тексты, такие как *Fractals, Visualization and J* [7], *Exploring Math* [8], и *Concrete Math Companion* [14] используют язык в различных областях.

В) Прилагаемый словарь языка **J** представляет из себя полный и лаконичный словарь с описанием грамматики.

Г) *J Phrases* [9] содержит примеры написанных программ. Практически каждая затронутая там тема содержит задачи различного уровня сложности.

Д) Слова имеют значимую структуру. Например: `+:` , `-:` , `*:` и `%:` означают: *удвоить*, *пополам*, *возвести в квадрат*, и *взять квадратный корень*. Кроме того, новичок может присвоить и пользоваться легко запоминаемыми именами на своем родном языке, как например: `sqrt=:%:` , `entier=:<.` (по-французски), `sin=:1&o.` и `SIN=:1&o.@(%&180@o.)` (для синуса угла, заданного в

градусах).

Далее мы будем вводить и использовать новые примитивы почти без обсуждения, предполагая, что читатель самостоятельно исследует их на компьютере, справится со словарем, чтобы уточнить значение, или, возможно, догадается о нем из структуры. Например, вид слова `o.` напоминает круг. Выше оно было использовано диадно для определения синуса (одной из *круговых* функций), и монадно для функции *умножить на пи*, тоесть длины *круга*, если аргументом является его диаметр.

Для чтения программ вслух может быть полезным использование имен (или их сокращений) самих символов, как в:

`<` Левая у(гловая скобка)	`/` Косая	`&` Амперсанд	`%` Процент
`[` Левая к(вадратная скобка)	`\` Обратная (косая)	`@` Поверх	`;` Точка с запятой
`{` Левая ф(игурная скобка)	`|` Палка	`^` Птичка	`~` Тильда
`(` Левая с(кобка)	`_` Подчерк	`` ` `` Обратный апостроф	`*` Звездочка

Упражнения

9.1 Поэкспериментируйте с новой версией программы `MAP` из Упражнения 7.1, заменив диаду *меньшее* (`<.`) на *остаток* от деления (`|`), как в `M=:map@(6&|)`. Сравните результаты с результатами `MAP`.

9.2 Поэкспериментируйте с программами `sin` и `SIN` , определенными в этом разделе.

9.3 Напишите программы с использованием других примитивов из Разговорника в конце этой книги.

9.4 Обновите таблицу обозначений, подготовленную в Упражнении 2.2.

10. Организация Сессии

Помнить все имена, присвоенные глаголам и существительным во время длинной сессии, может быть непросто. *Внешний* союз `!:` (детально описанный в Приложении А) дает возможность просмотреть и стереть их. Например:

```
   b=: 3* a=: i. 6
   sum=: +/
   tri=: sum\ a
   names=: 4!:1

   names 0
+-+-+---+
|a|b|tri|
+-+-+---+

   names 3
+-----+---+
|names|sum|
+-----+---+

   erase=: 4!:55
   erase <'tri'
1

   names 0 3
+-+-+-----+-----+---+
|a|b|erase|names|sum|
+-+-+-----+-----+---+

   erase names 0
1 1

   names 0 3
+-----+-----+---+
|erase|names|sum|
+-----+-----+---+
```

Выпадающие меню позволяют сохранить, восстановить и распечатать сессию. Кроме того, с их помощью можно открывать *окна сценариев*, служащие для ввода и редактирования любого количества предложений и позволяющие *выполнить* их все сразу, будто они введены подряд в обычном окне непосредственного исполнения.

Сценарии и окно исполнения можно расположить впритык, чтобы эффект от выполнения сценариев был виден непосредственно.

В файлах подсказки (меню *Help*) находится детальное описание меню и дополнительных команд (определенных в файле *profile.ijs*) для организации сессий. Чтобы сделать эти команды доступными,

нужно выполнить *profile.ijs* либо явно (при помощи меню), либо (чтобы он был выполнен автоматически в начале сессии), включив его имя в *командную строку*, как описано в руководстве пользователя.

Упражнения

10.1 Введите и исследуйте с программы, определенные в этом разделе.

10.2 Наберите предложение `+/ 2 3 5 * i. 3` и нажмите Ввод, чтобы его выполнить.

10.3 Используйте стрелку вверх (из клавиш управления курсором) для перемещения курсора к строке, введенной в Упражнении 10.2, а потом:

Нажмите Ввод, чтобы перенести строку вниз, в область ввода.

Используя стрелку влево, переместите курсор назад к символу * , и сотрите его клавишей Забой (часто обозначаемой стрелкой влево, на алфавитной части клавиатуры).

Введите `-` для замены умножения на вычитание, и нажмите Ввод, чтобы выполнить измененное предложение.

10.4 Используя клавиши управления курсором, переведите его в позицию слева от `i` , выражения, выполненного в Упражнении 10.3. Потом, удерживая клавишу Ctrl, нажмите F1, чтобы посмотреть словарное определение примитива `i.`

10.5 Нажмите клавишу Esc, чтобы закрыть окно, вызванное в Упражнении 10.4. Потом переместите курсор налево, так, что он отделен от начала строки одним или несколькими пробелами. Нажмите Ctrl-F1, чтобы увидеть индивидуально упакованные слова в предложении.

11. Степень и Обратные Глаголы

Союз *в степени* (`^:`) подобен функции возведения в степень (^). Например:

```
   ]a=: 10^ b=: i.5
1 10 100 1000 10000

   b
0 1 2 3 4

   %:a
1 3.16228 10 31.6228 100

   %: %: a
1 1.77828 3.16228 5.62341 10

   %: ^: 2 a
1 1.77828 3.16228 5.62341 10

   %: ^: 3 a
1 1.33352 1.77828 2.37137 3.16228

   %: ^: b a
1      10     100    1000   10000
1 3.16228      10 31.6228     100
1 1.77828 3.16228 5.62341      10
1 1.33352 1.77828 2.37137 3.16228
1 1.15478 1.33352 1.53993 1.77828

   (cos=: 2&o.) ^: b d=:1
1 0.540302 0.857553 0.65429 0.79348

   ] y=: cos ^: _ d
0.739085

   y=cos y
1
```

Как видим, последовательные применения `cos` дают результаты, сходящиеся к некоторому предельному значению; бесконечная степень (`cos ^: _`) вычисляет этот предел.

Правый аргумент `_1` производит *обратную* функцию. Тоесть:

```
   %: ^: _1 b
0 1 4 9 16

   *: b
0 1 4 9 16

   %: ^: (-b) b
0 1     2         3         4
```

```
0 1     4         9        16
0 1    16        81       256
0 1   256      6561     65536
0 1 65536 4.30467e7 4.29497e9
```

Упражнения

11.1 Функция возведения в квадрат *: обратна взятию квадратного корня %:. Таким образом, %:^:_1 эквивалентно *: . Найдите (среди примитивов, перечисленных в Разговорнике) и исследуйте другие пары обратных функций.

12. Чтение и Письмо

При изучении нового языка, полезным упражнением являются переводы в обе стороны на другой, известный. Вот например:

Закройте правую половину страницы и попробуйте (всерьез) перевести предложения, записанные **слева** на русский. Тоесть, воспроизведите кратко по-русски действие глаголов, определяемых каждым предложением. Используйте любую доступную помощь, включая словарь и эксперименты на компьютере:

`f1=: <:`	Вычесть 1 (монада); Меньше или равно
`f2=: f1&9`	Меньше или равно 9
`f3=: f2 *. >:&2`	Проверка на попадание в интервал от 2 до 9 (включительно)
`f4=: f3 *. <. = ]`	От 2 до 9 и целое
`f5=: f3 +. <. = ]`	От 2 до 9 или целое
`g1=: %&1.8`	Разделить на 1.8
`g2=: g1^:_1`	Умножить на 1.8
`g3=: -&32`	Вычесть 32
`g4=: g3^:_1`	Прибавить 32
`g5=: g1@g3`	В шкалу Цельсия из шкалы Фаренгейта
`g6=: g5^:_1`	Фаренгейт из Цельсия
`h1=: >./`	Максимум в списке (монада)
`h2=: h1-<./`	Разброс значений. Попр. `h2 b` с параболой: `b=:(-&2 * -&3) -:i.12`
`h3=: h1@]-i.@[*h2@]%<:@[`	На сетке. Попробуйте `10 h3 b`
`h4=: h3 <:/ ]`	Гистограмма. Попробуйте `10 h4 b`
`h5=: {&' *' @ h4`	Гистограмма. Попробуйте `10 h5 b`

После ввода предыдущих определений, введите имя каждого глагола без аргументов, чтобы увидеть его определение и научиться его интерпретировать. Введите перед началом `9!:3 (2 4 5)` для получения нескольких форм отображения.

Закрыв левую сторону страницы, попробуйте перевести определения на русском **справа** обратно на **J**.

Упражнения

12.1 Эти упражнения сгруппированы по темам и организованы как раздел. Программы нужно сначала прочитать, потом записать самостоятельно. Читатель, хорошо знакомый с какой либо из предложенных тем, может начать сразу с записи.

A. Свойства чисел

`pn=: >:@i.`	Целые положительные числа (т.е. `pn 9`)
`rt=: pn |/ pn`	Таблица остатков
`dt=: 0&=@rt`	Таблица делимости
`nd=: +/@dt`	Количество делителей
`prt=: 2&=@nd`	Проверка на простое число
`prsel=: prt # pn`	Выбрать простые
`N=: >:@pn`	Целые числа, начиная с единицы
`rtt=: ,@(N */ N)`	Разобранная таблица умножения
`aprt=: -.@(N e. rtt)`	Альтерн. проверка и выборка (простых нет в т. умножения для `N`)
`apsel=: aprt # N`	
`pfac=: q:`	Разложение на простые множители
`first=: p:@i.`	Первые простые числа

B. Декартова Геометрия

Сделайте эксперименты с вектором (или *точкой*) `p=: 3 4` и треугольником, представленным таблицей `tri=: 3 2$ 3 4 6 5 7 2`

`L=: %:@:(+/)@:*:"1`	Длина вектора (См. *с рангом* в словаре, или в Разделе 20 Введения)
`LR=:L"1`	Длина векторов, представленных строками в таблице
`disp=: ] - 1&|.`	Смещение от строки к строке
`LS=: LR@disp`	Длины сторон фигуры
`sp=: -:@(+/)@LS`	Полу-периметр (попробуйте `sp tri`)
`H=: %:@(*/)@(sp,sp-LS)`	Формула Герона для площади
`det=: -/ . *`	Определитель (См. словарь "По Минорам")
`SA=: det@(,.&0.5)`	Площадь со знаком. Попробуйте `SA@|.`
`sa=:det@(],.%@!@<:@#)`	Обобщенный обьем со знаком; попробуйте для тетраэдра

tet=:?4 3$9 и для треугольника tri .

13. Форматирование

Числовую таблицу, такую как:

```
   ]t=:(i.4 5)%3
      0 0.333333 0.666667       1 1.33333
1.66667        2  2.33333 2.66667       3
3.33333  3.66667        4 4.33333 4.66667
      5  5.33333  5.66667       6 6.33333
```

можно отообразить в более удобном виде путем *форматирования*, позволяющего задать ширину каждого столбца и количество цифр после десятичной точки. Например:

```
   ]f=: 6j2 ": t
  0.00  0.33  0.67  1.00  1.33
  1.67  2.00  2.33  2.67  3.00
  3.33  3.67  4.00  4.33  4.67
  5.00  5.33  5.67  6.00  6.33
```

Действительная часть левого аргумента функции форматирования указывает ширину столбца, а мнимая часть -- количество цифр после десятичной точки.

Хоть форматированная таблица *выглядит* почти как исходная, она не числовая, а *текстовая*, состоящая из символов алфавита. Например:

```
   $t
4 5

   $f
4 30

   +/t
10 11.3333 12.6667 14 15.3333

   +/f
|domain error
|       +/f
```

Глагол *сделать* или *выполнить* (`".`), в применении к такой текстовой таблице, возвращает соответствующую таблицу чисел:

```
   ". f
   0 0.33 0.67    1 1.33
1.67    2 2.33 2.67    3
3.33 3.67    4 4.33 4.67
   5 5.33 5.67    6 6.33
```

```
   +/ ". f
10 11.33 12.67 14 15.33
```

Упражнения

13.1 Исследуйте следующие выражения, использующие программы из Раздела 12:

```
5j2 ": d=: %: i.12
5j2 ":,.d
fc=: 5j2&":@,.
fc d
20 (fc@h3 ,. h5) d
20 (fc@h3 ,. '|'&,.@h5) d
plot=: fc@h3,.'|'&,.@h5
20 plot d
```

14. Разбиения

Функция `sum=: +/` обрабатывает свой правый аргумент целиком и полностью. Для вычисления *частичных сумм* или *под-сумм* ее необходимо применить к каждому префиксу аргумента. Например:

```
   sum=: +/
   a=: 1 2 3 4 5 6
   (sum a) ; (sum\ a)
+--+--------------+
|21|1 3 6 10 15 21|
+--+--------------+
```

Символ `\` обозначает наречие *префиксно,* применяющее свой аргумент (в данном случае `sum`) к каждому префиксу аргумента результирующего глагола. Наречие `\.` применяет глаголы подобным образом, но к суффиксам:

```
   sum\. a
21 20 18 15 11 6
```

Поскольку монада `<` просто *упаковывает* свои аргументы, глаголы `<\` и `<\.` ясно показывают осуществляемые разбиения. Например:

```
   <1 2 3
+-----+
|1 2 3|
+-----+

   (<1),(<1 2),(<1 2 3)
+-+---+-----+
|1|1 2|1 2 3|
+-+---+-----+

   <\ a
+-+---+-----+-------+---------+-----------+
|1|1 2|1 2 3|1 2 3 4|1 2 3 4 5|1 2 3 4 5 6|
+-+---+-----+-------+---------+-----------+

   <\. a
+-----------+---------+-------+-----+---+-+
|1 2 3 4 5 6|2 3 4 5 6|3 4 5 6|4 5 6|5 6|6|
+-----------+---------+-------+-----+---+-+
```

Наречие *диагонально* `/.` разбивает *таблицу* на диагонали. Тоесть:

```
   </. t=: 1 2 1 */ 1 3 3 1
+-+---+-----+-----+---+-+
|1|3 2|3 6 1|1 6 3|2 3|1|
+-+---+-----+-----+---+-+

   t ; (sum/. t) ; (10 #. sum/. t) ; (121*1331)
```

```
+-------+-------------+------+------+
|1 3 3 1|1 5 10 10 5 1|161051|161051|
|2 6 6 2|             |      |      |
|1 3 3 1|             |      |      |
+-------+-------------+------+------+
```

Упражнения

14.1 Подобно `sum=:+/\` , определите программы, вычисляющие частичные произведения, частичные максимумы и частичные минимумы.

14.1 Поработайте со следующими программами и комментариями, как в Разделе 12 (тоесть используйте их для упражнений в чтении и письме). Поэкспериментируйте с выражениями `c pol x` , `c pp d` и `(c pp d) pol x` , используя `c=:1 3 3 1` , `d=:1 2 1` и `x=:i.5` . См. словарь или Раздел 20 относительно использовании ранга:

`pol=: +/@([*]^i.@#@[)"1 0`	Многочлен
`pp=: +//.@(*/)`	Произведение многочленов
`pp11=: 1 1&pp`	Произведение многочлена на 1 1
`pp11 d`	
`pp11^:5 (1)`	
`ps=: +/@,:`	Сумма многочленов

15. Определение Наречий

Так же как существительным и глаголам, наречиям можно присваивать имена:

```
   a=:1 2 3 4 5
   prefix=: \
   < prefix 'abcdefg'
+-+--+---+----+-----+------+-------+
|a|ab|abc|abcd|abcde|abcdef|abcdefg|
+-+--+---+----+-----+------+-------+

   +/ prefix a
1 3 6 10 15
```

Новые наречия можно построить в виде цепочки наречий (таких как /\) или союза с одним аргументом. Такие наречия можно *определить*, присвоив им имена. Тоесть:

```
   IP=: /\                    NB. Между префиксами
   + IP a
1 3 6 10 15

   with3=: &3
   % with3 a
0.333333 0.666667 1 1.33333 1.66667

   ^ with3 a
1 8 27 64 125

   I=: ^: _1                  NB. Наречие обращения функции
   *: I a
1 1.41421 1.73205 2 2.23607

   + IP I 1 3 6 10 15
1 2 3 4 5

   ten=: 10&
   ^. ten 5 10 20 100
0.69897 1 1.30103 2

   #. ten 3 6 5
365

   from=: -~
   into=: %~
   10 into 17 18 19
1.7 1.8 1.9

   10 from 17 18 19
7 8 9

   i=: "_1               NB. Применить к элементам
```

```
   {. i i. 3 4
0 4 8
```

Упражнения

15.1 Исследуйте поведение наречий `pow=: ^&` и `log=: &^.`

15.2 Определите предполагаемый результат следующих выражений, и проверьте себя, введя их:

`+/~ i=: i. 6`	Таблица сложения
`ft=: /~`	Наречие "таблица функции"
`+ ft i`	Таблица сложения
`! ft i`	Биномиальные коэффициенты
`inv=: ^:_1`	Наречие обращения
`sub3=: 3&+ inv`	Функция "вычесть 3"
`sub3 i`	

16. Выделение Слов

Интерпретация написанного предложения на русском начинается с разбиения его на слова. Основой для него служат пробелы, разбивающие предложение на отдельные единицы письменного текста, но процесс усложняется необходимостью принять во внимание дефисы и знаки препинания: *было*, *Петров* и *Татаринов-Григорьев* рассматриваются как отдельные единицы, но *однако,* такой единицей не является (поскольку запятая -- это тоже независимая единица).

Следующие текстовые списки представляют собой предложения на **J**, которые можно выполнить глаголом *сделать* (или *выполнить*) `". :`

```
   m=: '3 %: y'
   d=: 'x %: y'
   x=: 4
   y=: 27 4096

   ". m
3 16

   do=: ".
   do d
2.27951 8
```

В языке **J** правила разбиения текста на слова определены в Главе I словаря. Кроме того, в применении к текстовому списку, содержащему предложение на **J**, функция *упаковки слов* `;:` производит упакованный список его слов:

```
   ;: m
+-+--+-+
|3|%:|y|
+-+--+-+

   words=: ;:
   words d
+-+--+-+
|x|%:|y|
+-+--+-+
```

Правила образования слов языка **J** неплохо работают и для фраз на естественных языках:

```
   words p=: 'Nobly, nobly, Cape St. Vincent'
+-----+-+-----+-+----+---+-------+
|Nobly|,|nobly|,|Cape|St.|Vincent|
+-----+-+-----+-+----+---+-------+

   >words p
```

```
Nobly
,
nobly
,
Cape
St.
Vincent
```

Упражнения

16.1 Выберите предложения из предыдущих упражнений (например, `pp=:+//.@(*/)`), заключите их в кавычки и понаблюдайте за разбиением их на слова при помощи `(;:)` .

16.2 Поместите курсор слева от строки, так, что он отделен от ее начала одним (или более) пробелом. Нажмите Ctrl-F1, чтобы увидеть упакованные по отдельности слова указанного предложения.

17. Имена и Отображенее

Вдобавок к (уже использованным) обычным именам, существует три дополнительных их класса:

1) `$:` обозначает ссылку на себя, позволяя определять рекурсивные глаголы, не называя их, как показано в Разделе 22.

2) Имена `x` и `y` используются в явных определениях, как обсуждается в Разделе 18. Они обозначают аргументы этих определений.

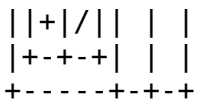
3) Имя (такое как `ab_cd_`), содержащее два подчерка, из которых один стоит в конце, называется *локативом*. В ситуации, когда из *пространства имен* `G` нужно сослаться на имя в другом пространстве имен `F` , локатив `pqr_F_` (с `F` в суффиксе) позволяет избежать конфликта с (возможно идентичными) именами в пространстве `G` . См. Раздел I Главы II для дополнительной информации.

Вид отображения, осуществляемого при вводе имени без аргументов, устанавливается `9!:3` , как описано в Приложении A. Например:

```
   mean=: +/ % #

   9!:3 (4)                    Древовидное отображение

   mean
  +- / --- +
--+- %
  +- #

   9!:3 (5)                    Линейное отображение

   mean
+/ % #
```

Можно установить несколько видов отображения одновременно:

```
   9!:3 (5 4 2)

   mean

+/ % #
  +- / --- +
--+- %
  +- #
+-----+-+-+
|+-+-+|%|#|
```



Упражнения

17.1 Поэкспериментируйте с локативами.

18. Явные Определения

Предложения (выполненные для примера в предыдущем разделе о разбиении на слова):

```
   m=: '3 %: y'
   d=: 'x %: y'
```

могут быть использованы с союзом *явно* : для производства глагола:

```
   script=: 0 : 0
3 %: y
:
x %: y
)

   roots=: 3 : script
   roots 27 4096
3 16

   4 roots 27 4096
2.27951 8
```

Наречия и союзы тоже можно определять явно. Например:

```
   table=: 1 : 0
[ by ] over x/                      by   over    _______ 3
)

   2 3 5 ^ table 0 1 2 3
+-+----------+
| |0 1  2   3|
+-+----------+
|2|1 2  4   8|
|3|1 3  9  27|
|5|1 5 25 125|
+-+----------+
```

Кроме того, в явных определениях можно использовать управляющие конструкции. Например:

```
f=: 3 : 0
if. y<0
  do. *:y
  else. %:y
end.
)

   f"0 (_4 4)
16 2
```

```
factorial=: 3 : 0
a=. 1
while. y>1
do. a=. a*y
y=. y-1 end. a
)

   factorial"0 i. 6
1 1 2 6 24 120
```

Упражнения

18.1 Поэкспериментируйте и отобразите функции `roots=: 3 : script` и `13 : script` (они эквивалентны).

18.2 См. обсуждение управляющих конструкций в словаре. Используйте их для определения дальнейших глаголов.

18.3 Поэкспериментируйте с выражениями вида `! d b=: i.7` , определив наречие `d` как:

```
d=: 1 : 0
+:@x
)
```

18.4 Используя программу `pol` из Упражнения 14.2, проведите следующие эксперименты и прокомментируйте их результаты:

```
g=: 11 7 5 3 2 & pol
e=: 11 0 5 0 2 & pol
o=:  0 7 0 3 0 & pol
(g = e + o) b=: i.6
(e = e@-) b
(o = -@o@-) b
```

Ответ: Функция `g` есть сумма функций `e` и `o` . Более того, функция `e` четная (ее график симметричен относительно вертикальной оси), а `o` -- нечетная (ее график симметричен относительно начала координат).

18.5 Просмотрите Раздел H Главы II и используйте сценарии для написания других явных определений.

18.6 Введите следующее явное определение наречия `even` (четно) и проведите с ним эксперименты, используя функции, определенные в предыдущем упражнении:

```
even=: 1 : 0
-:@(x f. + x f.@-)
)
ge=: g even
(e = ge) b
(e = e even) b
```

18.7 Определите наречие `odd` (нечетно) и используйте его в следующих экспериментах:

```
exp=: ^
sinh=: 5&o.
cosh=: 6&o.
(sinh = exp odd) b
(sinh = exp .: -) b                                    "       " .: -
(cosh = exp even) b
```

```
(exp = exp even + exp odd) b
```

18.8 Следующие эксперименты используют комплексные числа. Тем, кто с ними не знаком, вероятно, следует эти эксперименты пропустить:

```
sin=: 1&o.
cos=: 2&o.
(cos = ^@j. even) b
(j.@sin = ^@j. odd) b
```

19. Неявные Эквиваленты

Глаголы можно определять как явно, так и неявно. В случае явного определения, состоящего из одного предложения, наречие `13 :` позволяет получить эквивалентное неявное определение, как показано ниже (вначале введите `9!:3 ] 2 5` , чтобы видеть коробочное и линейное отображение глаголов):

```
   s=: 0 : 0
(+/y) % (#y)
)

   mean=: 3 : s
   MEAN=: 13 : s

   mean                          MEAN
+-+-+------------+          +-----+-+-+
|3|:|(+/y) % (#y)|          |+-+-+|%|#|
+-+-+------------+          ||+|/|| | |
3 : '(+/y) % (#y)'          |+-+-+| | |
                            +-----+-+-+
                            +/ % #
```

Опытным программистам явные определения, вероятно, покажутся более знакомыми чем неявные. Перевод, производимый наречием `13 :` , может быть полезен при изучении неявного программирования.

Упражнения

19.1 Используйте отображение неявного определения `MEAN` , чтобы определить эквивалентную функцию с именем `M` .

Ответ: `M=: +/ % #`

20. Ранг

Размерность ($), количество элементов (#) и ранг (#@$) легко проиллюстрировать на примере существительного report (его можно, например, считать отчетом, покрывающим два года, состоящие из четырех кварталов, по три месяца в каждом):

```
   ]report=: i. 2 4 3
 0  1  2
 3  4  5
 6  7  8
 9 10 11

12 13 14
15 16 17
18 19 20
21 22 23

   ($ ; # ; #@$) report        Размерность, число элементов, ранг
+-----+-+-+
|2 4 3|2|3|
+-----+-+-+
```

Последние *k* измерений составляют *k-ячейку* существительного; 0-ячейками существительного report являются атомы (например: 4 и 14), 1-ячейками являются трех-элементные квартальные отчеты, а 2-ячейками (*главными ячейками* или *элементами*) два годовых отчета размерности четыре на три.

Союз *с рангом* " используется в выражении f"k для применения функции f к каждой из k-ячеек аргумента. Например:

```
   ,report
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

   ,"2 report
 0  1  2  3  4  5  6  7  8  9 10 11
12 13 14 15 16 17 18 19 20 21 22 23

   <@i. s=: 2 5
+---------+
|0 1 2 3 4|
|5 6 7 8 9|
+---------+
   <@i."0 s
+---+---------+
|0 1|0 1 2 3 4|
+---+---------+
```

Для диад можно указать левый и правый ранги. Тоесть:

```
   10 11 12 (,"0 1 ; ,"1 1 ; ,"1) 0 1 2
+--------+-------------+-------------+
|10 0 1 2|10 11 12 0 1 2|10 11 12 0 1 2|
|11 0 1 2|              |              |
|12 0 1 2|              |              |
+--------+-------------+-------------+
```

Наречие *свойства* `b.` очень полезно для анализа функций (или выражений их определяющих) на предмет ранга. Например:

```
   (# b. 0) ; (+/\ b. 0) ; (+/\ % #) b. 0
+-----+-----+-----+
|_ 1 _|_ 0 _|_ _ _|
+-----+-----+-----+
```

Упражнения

20.1 Посмотрите на результаты следующих монад, произведенных союзом *с рангом*, и прокомментируйте их:

```
a=: i. 3 4 5
<"0 a
<"1 a
<"2 a
<"3 a
< a
<"_1 a
<"_2 a
mean=: +/ % #
mean a
mean"1 a
mean"2 a
```

Ответ: `<"k` применяет `<` к каждой ячейке ранга `k` , причем `<"(#$a) a` эквивалентно `<a` . Отрицательное значение `k` обозначает *дополнительный* ранг, соответствующий обычному рангу в виде разности ранга аргумента `a` и `|k` .

20.2 Установите связь между союзами `@` (*Поверх*) и `@:` (*Над*) при помощи следующих экспериментов; сравните свои выводы с определениями в словаре:

```
(g=: <"2) a=: i. 3 4 5
|. @: g a
|. @ g a
|: @: (<"1) a
|: @ (<"1) a
```

Ответ: Функция `|. @: g` имеет бесконечный ранг, а значит `|.` применяется к результату `g` целиком, меняя порядок элементов

в нем. С другой стороны, функция `f @ g` *наследует* ранг `g` , а значит `|.` в ней применяется индивидуально к каждому атому результата `g` , не производя никакого эффекта.

20.3 Проведите следующие эксперименты и прокомментируйте использование союза *с рангом* к диадам:

```
b=: 'ABC'
c=: 3 5 $ 'abcdefghijklmno'
c
b,c
b ,"0 1 c
b ,"1 1 c
b ,"1 c
```

21. Герундий и Сообразно

В английском, *герундием* называется существительное с окончанием -ing, имеющее одновременно свойства глагола (как, например, *cooking* в выражении *the art of cooking*). *Сообразно* (например плану) подразумевает выбор из списка элементов, предписывающих некоторые действия. Союз *и/или* `` ` `` в применении к двум глаголам производит герундий, элементы которого могут быть потом выполнены по выбору при помощи союза *сообразно* `@.` , правый глагольный аргумент которого производит индексы, выбирающие элементы герундия к исполнению. Например:

```
   g=: +`^
   a=:<
   2 a 3
1

   2 g@.a 3
8

   3 g@.a 2
5

   +:`-:`*:`%: @. (4&|@<.)"0 i. 10
0 0.5 4 1.73205 8 2.5 36 2.64575 16 4.5
```

Произведенный `g@.a` глагол часто называется *выбор* или *оператор case*, поскольку он выбирает для исполнения один из "вариантов" герундия.

Применение к герундию наречия *между* `/` аналогично его применению к глаголу. Например:

```
   c=:3 [ x=: 4 [ power=: _1
   g/ c,x,power
3.25

   3+x^_1
3.25
```

Элементы герундия повторяются циклически, сколько требуется. Например:

```
   +`*/1,x,3,x,3,x,1
125
```

Последнее предложение соответствует правилу Хорнера для эффективного вычисления многочлена с коэффициентами `1 3 3 1` и аргументом `x` .

Упражнения

21.1 Определите функцию `f` , такую что `(x=: 4) f c=: 1 3 3 1` дает результат, применимый потом в качестве аргумента для `+`*/` в методе Хорнера.

Ответ: `f=: }.@,@,.`

22. Рекурсия

Факториал `!` целого числа `n` определяется как произведение `n` и факториала `n-1` , с граничным условием: факториал `0` равен `1` . Такое определение называется *рекурсивным*, поскольку функция определяется через вызов самой себя (function recurs).

Рекурсивное определение можно дать при помощи союза *сообразно*, выбирающего вариант обращения к себе до тех пор, пока не выполнено граничное условие. Например:

```
   factorial=: 1:`()*factorial@<:) @. *
   factorial "0 i.6
1 1 2 6 24 120
```

Заметьте, что `1:` обозначает постоянную функцию, результат которой всегда `1` .

В предложении `(sum=: +/) i.5` глаголу, определенному фразой `+/` , перед использованием присваивается имя, но в предложении `+/ i.5` он используется анонимно. В определении `factorial` присваивание имени было необходимо для того, чтобы сослаться на него внутри определения. Однако, слово `$:` позволяет *сослаться на себя* без присвоения имени, открывая возможность написания анонимных рекурсивных определений. Например:

```
   1:`()*$:@<:) @. * "0 i. 6
1 1 2 6 24 120
```

В задаче о Ханойской Башне, набор `n` дисков (каждый своего размера) должен быть перемещен со стержня A на стержень B, используя третий стержень C, при ограничении, что больший диск нельзя класть на меньший. Следующее, представляет из себя рекурсивное определение этого процесса:

```
   h=: b`(p,.q,.r)@.c
    c=: 1: < [
    b=: 2&,@[ $ ]
      p=: <:@[ h 1: A. ]
      q=: 1: h ]
      r=: <:@[ h 5: A. ]

   3 h x=: 'ABC'
AABACCA
BCCBABB

   0 1 2 3 4 <@h"0 1 x
++-+---+-------+---------------+
||A|AAC|AABACCA|AACABBAACCBCAAC|
```

```
||B|CBB|BCCBABB|CBBCACCBBAABCBB|
++-+---+-------+---------------+
```

Упражнения

22.1 Используйте следующее для упражнений по чтению и письму:

`f=:1:`(+//.@(,:~)@($:@<:))@.*`	Биномиальные коэффициенты
`<@f"0 i.6`	Упакованные бином. коэффицие
`g=:1:`((],+/@(_2&{.))@$:@<:)@.*`	Числа Фибоначчи

23. Итерация

Повторение процесса (или последовательность подобных процессов) называется *итерацией*. Большая часть итераций делается неявно, как в `*/b` , `a*/b`, и `a*b` ; явную итерацию можно осуществить при помощи союза *в степени* (`^:`), союза *сообразно* (`@.`) с *вызовом себя* (`$:`) и управляющих конструкций:

```
   (cos=: 2&o.) ^: (i.6) b=: 1
1 0.540302 0.857553 0.65429 0.79348 0.701369

   ]y=: cos^:_ b
0.739085

   y=cos y
1
```

Пример `cos^:_` иллюстрирует тот факт, что бесконечная итерация может иметь смысл (тоесть она оканчивается) если повторяемый процесс сходится.

Управляемая итерация процесса `p` осуществляется при помощи `p^:q` , где результат `q` определяет количество повторений `p` , которые будут сделаны. Полезны так же выражения вида `$:@p^:q` и `p^:q^:_`.

Если функция `f` непрерывна, а `f i` и `f j` отличаются знаком, то это значит, что между `i` и `j` находится *корень* `r` , такой, что `f r` равно нулю. Тогда говорят, что список `b=:i,j` *локализует* корень. Более точную локализацию можно осуществить, взяв среднее `b` вместе с точкой из `b` , применение `f` к которой отличается знаком от значения функции в средней точке. Тоесть:

```
   f=: %: - 4:                NB. Тестовая функция для примера

   root=: 3 : 0
m=. +/ % #
while. ~:/y
do.
if. ~:/ * f ({.,m) y
   do. y=. ({.,m) y
 else. y=. ({:,m) y
end.
end. m y
)

   b=: 1 32
   root b
16

   f b,root b
_3 1.65685 1.77636e_15
```

Упражнения

23.1 Используйте функцию `root` для нахождения корней различных функций, как например `f=: 6&-@!`

23.2 Поэкспериментируйте с функцией `fn=: +/\` (производящей, при повторном применении к `i.n` , фигурные числа), объясните поведение функции `fn^:(?@3:)`

24. Перестановки

Анаграммы являются известным примером *перестановок*:

```
   w=: 'STOP'
   3 2 0 1 { w
POST

   2 3 1 0 { w
OPTS

   3 0 2 1 { w
PSOT
```

Левые аргументы { выше, сами являются перестановками (списка i.4) ; они называются *векторами перестановок*, и используются для представления переставляющих функций в виде p&{ .

Если p вектор перестановки, то выражение p&C. выполняет перестановку так же как p&{ . Однако, в других случаях, функция *циклы* C. отличается от функции *взять* { . В частности, C. p дает *циклическое представление* перестановки p . Например:

```
   ]c=: C. p=: 2 4 0 1 3
+---+-----+
|2 0|4 3 1|
+---+-----+

   c C. 'ABCDE'                  C. c
CEABD                         2 4 0 1 3
```

Каждый упакованный элемент этого представления перечисляет список позиций, по которым происходит циклическое перемещение элементов при перестановке. В примере выше, элемент из позиции 3 перемещается в позицию 4 , элемент 1 перемещается в 3 , а элемент 4 в 1 .

Перестановке можно сопоставить ее индекс в таблице всех !n перестановок порядка n , перечисленных по возрастанию. Этот индекс называется *индексом перестановки* (*anagram index*). Соответствующую перестановку можно осуществить функцией A. , как показано ниже:

```
   1 A. 'ABCDE'                  A. 0 1 2 4 3
ABCED                         1

   (i.!3) A. i.3                 (i.!3) A. 'ABC'
0 1 2                         ABC
0 2 1                         ACB
1 0 2                         BAC
```

```
1 2 0                          BCA
2 0 1                          CAB
2 1 0                          CBA
```

Упражнения

24.1 Используйте следующее для упражнений по чтению и письму (в качес
исходных данных попробуйте `a=:'abcdef'` , `b=: i. 6` и `c=: i. 6 6`):

```
f=: 1&A.                       Переставить два последних элемента
g=: 3&A.                       Провернуть три последних элемента
h=: 5&A.                       три последних элемента в обратном поря
i=: <:@!@[ A. ]                k i a -- последние k элементов в обр. пор
```

24.2 Поэкспериментируйте со следующими (и другими подобными)
выражениями, для определения правила использования сокращенных
аргументов `C.` ; сравните свои выводы с определениями в словаре:

```
2 1 4 C. b=:i.6
(<2 1 4) C. b
(3 1;5 0) C. b
```

24.3 Напишите программу `ac` , производящую таблицу циклических
представлений всех перестановок заданного порядка, вызываемую,
например, как `ac 3` .

Ответ: `ac=: C.@(i.@! A. i.)`

25. Линейные Функции

Функция f называется *линейной*, если f(x+y) равно (f x)+(f y) для всех аргументов x и y . Например:

```
   f=: 3&|. @: +: @: |.
   ]x=: i.# y=:2 3 5 7 11
0 1 2 3 4

   x+y                          f x+y
2 4 7 10 15                  8 4 30 20 14

   (f x),:(f y)                 (f x)+(f y)
2 0  8  6  4                 8 4 30 20 14
6 4 22 14 10
```

Можно дать и эквивалентное определение: f линейна, если f@:+ и +&f эквивалентны. Например:

```
   x f@:+ y                     x +&f y
8 4 30 20 14                 8 4 30 20 14
```

Если f линейна, то f y можно выразить в виде *матричного произведения* mp&M y , где

```
   mp=: +/ . *
   M=: f I=: = i.#y             I единичная матрица

   mp&M y
6 4 22 14 10
   f y
6 4 22 14 10
```

Обратное тоже верно, если m -- любая квадратная матрица порядка #y , то m&mp является линейной функцией y , а если m обратима, то (%.m)&mp -- соответствующая обратная функция:

```
   x=: 1 2 3 [ y=: 2 3 5
   ]m=: ?. 3 3$9
3 8 8
4 2 0
2 7 4
   ]n=: %. m
0.0909091   0.272727 _0.181818
_0.181818 _0.0454545  0.363636
 0.272727 _0.0568182 _0.295455

   g=: mp&m
   h=: mp&n

   x g@:+ y
```

```
45 90 56

   x +&g y
45 90 56
    g h y
2 3 5
```

Упражнения

25.1 Для каждой из перечисленных функций найдите матрицу `M`, такую что `M (mp=: +/ . *) N` эквивалентно исходной функции в применении к матрице `N`. Проверьте для `N=: i. 6 6`

```
|.
-
+:
(4&*-2&*@|.)
2&A.
```

26. Обращение и Преобразования

Результат `f^:_1` называется *обращением* функции `f` ; если `f=: g :. h`, то ее обращением является `h`, иначе обращение -- глагол, обратный к `f`. Обратные глаголы определены для более чем 25 примитивов (включая квадратный корень, как показано в Разделе 11). Обращение выполняется автоматически и для многих диад с прицепом, таких как `-&3` , `10&^.` и `2&o.` . Более того, `u@v^:_1` есть `(v^:_1)@(u^:_1)` . Например:

```
   fFc=: (32&+)@(*&1.8)
   ]b=:fFc _40 0 100
_40 32 212

   cFf=: fFc^:_1
   cFf b
_40 0 100
```

Результатом фразы `f &. g` является глагол `(g^:_1)@(f & g)` . Он может быть рассмотрен как применение "основной" функции *под преобразованием* (выполняемым глаголом `g` до операции, с восстановлением, путем обратного преобразования, после нее). Например:

```
   b=: 0 0 1 0 1 0 1 1 0 0 0
   sup=: </\                    Подавить единицы после первой
   sup b
0 0 1 0 0 0 0 0 0 0 0

   |. sup |. b                  Подавить единицы перед последней
0 0 0 0 0 0 0 1 0 0 0

   sup&.|. b
0 0 0 0 0 0 0 1 0 0 0

   3 +&.^. 4                    Умножить, путем взятия экспоненты
12                              от суммы логарифмов

   (^.3)+(^.4)
2.48491

   ^ (^.3)+(^.4)
12

   ]c=: 1 2 3;4 5;6 7 8
+-----+---+-----+
|1 2 3|4 5|6 7 8|
+-----+---+-----+

   |.&.> c                      Распаковать, перевернуть, и упаковать
+-----+---+-----+
```

```
|3 2 1|5 4|8 7 6|
+-----+---+-----+
```

Упражнения

26.1 Используйте следующее для упражнений по чтению и письму. Попроб
с аргументами `a=: 2 3 5 7`, `b=: 1 2 3 4` и `c=: <@i."0 i. 3 4`:

`f=: +&.^.`	Умножение сложением нат. логарифмов
`g=: +&.(10&^.)`	Умножение сложением логарифмов по ос
`h=: *&.^`	Сложение путем умножения
`i=: |.&.>`	Реверс внутри каждой упаковки
`j=: +/&.>`	Суммировать внутри каждой упаковки
`k=: +/&>`	то-же, но оставив в распакованном виде

27. Тождественные Функции и Нейтральные Элементы

Монады `0&+` и `1&*` являются *тождественными* функциями, а `0` и `1` называются *тождественными элементами* или *нейтралами* диад `+` и `*`, соответственно. Вставка между элементами пустого списка дает нейтрал для вставляемой диады. Например:

```
   +/ i.0         +/''           +/0{. 2 3 5
0              0              0

   */i.0          */''           */0{. 2 3 5
1              1              1
```

Эти свойства полезны при разбиении списков; они гарантируют, что некоторые очевидные соотношения выполняются даже когда один из элементов разбиения пустой. Например:

```
   +/ a=: 2 3 5 7 11
28

   (+/4{.a)+(+/4}.a)
28

   (+/0{.a)+(+/0}.a)
28

   */a
2310

   (*/4{.a)*(*/4}.a)
2310

   (*/0{.a)*(*/0}.a)
2310
```

Тождественные функции и другие основные свойства функций (такие как *ранг*) можно получить наречием `b.`, как показано ниже:

```
   ^ b. _1      Обратный
^.

   ^ b. 0       Ранги
_ 0 0

   ^ b. 1       Тождественная функция
$&1@(}.@$)
```

Упражнения

27.1 Попробуйте предсказать и проверьте результаты следующих выражений:

```
*/''
<./''
>./''
>./0 4 4 $ 0
+/ . */ 0 4 4 $ 0
1 2 3 4 +&.^./ 5 6 7 8
```

27.2 Исследуйте диаду `{@;` и назовите термин, используемый для нее в математике.

Ответ: Декартово произведение (Cartesian product)

27.3 Проверьте предположение, что монады (`%:@~. +/ . * =`) и `%:` эквивалентны. Чем может быть полезна первая, в применении к спискам типа `1 4 1 4 2` , содержащим одинаковые элементы ?

Ответ: Функция `%:` (которая может быть сложной для вычисления) применяется только к уникальным элементам аргумента, выбранным функцией *построить множество* `~.` .

27.4 Прокомментируйте следующие эксперименты перед чтением комментариев справа:

```
a=: 2 3 5 [ b=: 1 2 4
a (f=: *:@+) b          Квадрат суммы
a (g=: +&*: + +:@*) b   Сумма квадратов плюс удвоенное
                        произведение
a (f=g) b               Выражение тождественности функций
a (f-:g) b              f и g представляют собой тавтологию
                        (результат
taut=: f-:g             которойвсегда "истина", тоесть 1).
```

27.5 Фраза типа `f-:g` может быть тавтологией только для диадного случая, только для монадного или для обоих. Используйте следующие тавтологии для упражнений в чтении и письме, отмечая область применимости (только диада, и т.д.):

```
t1=: >: -: > +. =       (диада) Примитив >: идентичен
                        больше или равно
t2=: <. -: -@>.&-       (оба) Меньшее, есть большее с
                        обратным знаком от аргументов с
                        обр. знаком; Пол есть минус
```

	потолок от аргумента с обр. знаком.
`t3=: <. -: >.&.-`	Тоже, что и `t2` но использует *преобразование*
`t4=: *:@>: -: *: + +: + 1:`	(монада) Квадрат `a+1` есть квадрат `a` плюс два `a` плюс `1`
`t5=: *:@>: -: #.&1 2 1"0`	Тоже, что и `t4` используя многочлен
`t6=: ^&3@>: -: #.&1 3 3 1"0`	Как `t5` для куба
`bc=: i.@>: ! ]`	Биномиальные коэффициенты
`t7=: (>:@]^[) -: ()#.bc@[)"0`	Как `t6` с `k&t7` для `k`-й степени
`s=: 1&o.`	Синус
`c=: 2&o.`	Косинус
`t8=: s@+-:(s@[*c@])+(c@[*s@])`	(диада) Сложение и вычитание
`t9=: s@--:(s@[*c@])-(c@[*s@])`	Формулы для синуса
`det=: -/ . *`	Определитель
`perm=: +/ . *`	Перманент
`sct=: 1 2&o."0@(,"0)`	Таблицы синуса и косинуса
`t10=: s@- -: det@sct`	Как `t9` , но используя определитель таблицы sin и cos
`t11=: s@+ -: perm@sct`	Как `t8` , но используя перманент
`S=: 5&o.`	Гиперболический синус
`C=: 6&o.`	Гиперболический косинус
`SCT=: 5 6&o."0@(,"0)`	таблица Sinh и Cosh
`t12=: S@+ -: perm@SCT`	Теорема сложения для sinh
`SINH=: ^ .: -`	Нечетная часть экспоненты
`COSH=: ^ .. -`	Четная часть экспоненты
`t13=: SINH -: S`	Sinh есть нечетная часть экспоненты
`t14=: COSH -: C`	Cosh есть четная часть экспоненты
`sine=: ^&.j. .: -`	Sin есть нечетная часть экспоненты
`t15=: sine -: s`	преобразуя умножением на `0j1`

27.6 Прокомментируйте следующие выражения, перед прочтением комментариев справа:

`g=: + > >.`	Проверить, если сумма превышает максимум.
`5 g 2`	Истинно для положительных аргументов,
`5 g _2 _1 0 1 2`	но в общем случае не истинно.
`f=: *.&(0&<)`	Проверка, если оба аргумента больше `0` .
`theorem=: f <: g`	Истинность результата `f` не превышает истинности `g` . Это можно выразить еще и как "если `f` (истинно) тогда `g` (истинно)" или, что "из `f` следует `g`".
`5 theorem _2 _1 0 1 2`	

28. Вторичные

Полезно дополнить, определенные языком, *примитивы* (или *первичные*), производными от них, *вторичными*, имена которых тоже можно легко распознать. Следующие примеры используют имена, начинающиеся с заглавной буквы:

`Ad=:`	`[ 0:}-@>:@$@]{.]`	Присоединить скаляр к диагонали
`Ai=:`	`>:@i.`	Целые, начиная с 1
`Area=:`	`[: Det ] ,. %@!@#"1`	Площадь (Обьем). Попробуйте `Area tet=:0,=i.3`
`Bc=:`	`i. !/ i.`	Биномиальные коэффициенты
`Bca=:`	`%.@Bc`	Биномиальные коэффициенты (чередующиеся)
`By=:`	`' '&;@,.@[ ,. ]`	По (формату; см. `Ta`)
`Cpa=:`	`]%.i.@#@]^/Ei@[`	Коэффициенты приближения многочленом
`CPA=:`	`1 : 'x@] %.i.@#@]^/Ei@['`	Коэффициенты приближения многочленом (наречие)
`Det=:`	`-/ . *`	Детерминант
`Dpc=:`	`1: }. ] * i.@#`	Коэффициенты производной многочлена
`D1=:`	`("0)(D.1)`	Производная (скалярная, первая)
`Ei=:`	`i.@(+*+0&=)`	Расширенный набор целых
`Epc=:`	`Bc@# X ]`	Раскрыть коэффициенты
`Ipc=:`	`0: , ] % Ai@#`	Коэффициенты интегрированного многочлена
`Inv=:`	`^:_1`	Обратная
`Id=:`	`=@i.`	Единичная матрица
`Mat=:`	`-: /:~`	Проверка на монотонное возрастание
`Mdt=:`	`-: \:~`	Проверка на монотонное убывание
`Mrg=:`	`+&$ {. ,@(|:@,:)`	Слияние
`Over=:`	`({.;}.)@":@,`	Поверх (формата; см `Ta`)
`Pad=:`	`2 : 'x%.]^/Ei@(y"_)'`	Приближение многочленом заданной степени
`Pp=:`	`+//.@(*/)`	Произведение коэффициента полинома
`Si=:`	`(Ei@+: - |) : (-/ i.)`	Целые, симметричные вокруг 0

`Span=:`	`2 : 'y"_ x\ ]'`	В окне, вычисленном применением левого аргумента
`S1=:`	`|:@|@(^!._1/~%.^/~)@i.`	Числа Стирлинга (1-го рода)
`S2=:`	`|:@ (^/~%.^!._1/~)@i.`	Числа Стирлинга (2-го рода)
`Ta=:`	`1 : '[By]Over x/'`	Наречие "таблично"
`Thr=:`	`] * 0.1&^@[ <: |@]`	Уровень для не-нулевых
`Tile=:`	`$@]{.[$~$@]+2:\|1:+$@]`	Покрытие (попр. `0 1 Tile i. 2 3 4`)
`X=:`	`+/ . *`	Умножить (матричное произведение)
`XA=:`	`-/ . *`	Умножить (чередуя знаки)

Упражнения

28.1 Введите определения вторичных (хотя бы тех, которые используются в этих упражнениях), а потом следующие выражения:

```
(Ai 2 3);(Ai 2 _3);(Ei 2 3);(Ei 2 _3)
(i.;i.@-;Ai;Ai@-;Ei;Ei@-) 4
(Si 4);(7 4 Si 4)
+Ta~@i. 4
(S1;S2) 7
(];X/;%/;%./;(%./%{.)) y=: (Bc ,: Bca) 5
(0 1&Cb;1 _1&Cb) i. 2 3 4
```

28.2 Проведите дальнейшие эксперименты с вторичными.

Примеры

В этом разделе приведены примеры использования **J** в различных областях. Их нужно просматривать в сочетании со словарем, сидя перед клавиатурой системы с **J** . Их так же можно использовать для *обучения* следующим образом:

- Прочитайте одно или два предложения и их результаты (на левой половине страницы), и попробуйте ясно описать его (их) действие.
- Введите вариации этих предложений и проверьте правильность своих описаний.
- Сверьтесь со словарем для подтверждения правильности понимания значения примитивов (таких как `i.` в применении к одному или двум аргументам). Используйте Разговорник, расположенный в конце книги, как предметный указатель к страницам словаря. В системе с графическим интерфейсом подсветите слово (напр. `/:`), последующее нажатие ctrl-F1 покажет его словарное описание.
- Попробуйте вводить сложные предложения *по частям*. Например, при изучении предложения `a.{~j+/i.26` (из Раздела Алфавит и Числа) попробуйте ввести `i. 26` и `j+/i.26` по отдельности.

1. Орфография

```
   phr=:'index=:a.i.''aA'''
   ;:phr
+-----+--+--+--+----+
|index|=:|a.|i.|'aA'|
+-----+--+--+--+----+

   $ ;:phr
5

   >;:phr
index
=:
a.
i.
'aA'

   (do=: ".) phr
97 65

   do 'abc =: 3 1 4 2'
3 1 4 2

   abc
3 1 4 2
```

2. Алфавит и Числа

```
   (a. {~ j +/ i.26) ; (j +/ i.6) ; (j=: a. i. 'aA') ; ($ a.)
+-------------------------+-------------------+-----+---+
|abcdefghijklmnopqrstuvwxyz|97 98 99 100 101 102|97 65|256|
|ABCDEFGHIJKLMNOPQRSTUVWXYZ|65 66 67  68  69  70|     |   |
+-------------------------+-------------------+-----+---+

   1 2 3{ t=: 8 32$a.         NB. Собственно, алфавит
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

В таблице t алфавит разделен на восемь строк, но ее полное отображение будет выглядеть странно из-за эффекта, производимого различными управляющими символами (такими как возврат каретки). Отображение упаковки <t более предсказуемо, поскольку при таком отображении управляющие символы заменяются пробелами.

```
   i. 2 5                  NB. Таблица целых
0 1 2 3 4
5 6 7 8 9

   r=: 0j1 _1 0j_1 1       NB. Квадратные корни из 1 и минус 1
   + r                     NB. (Комплексное) сопряжение
0j_1 _1 0j1 1

   r * +r
1 1 1 1

   r */ r                  NB. Таблица умножения корней из +/- единицы
  _1 0j_1    1  0j1
0j_1    1  0j1   _1
   1  0j1   _1 0j_1
 0j1   _1 0j_1    1

   ! 45x                   NB. "x" дает неограниченную точность
119622220865480194561963161495657715064383733760000000000
```

3. Грамматика

```
   fahrenheit =: 50
   (fahrenheit - 32) * 5%9
10

   prices =: 3 1 4 2
   orders =: 2 0 2 1
   orders * prices
6 0 8 2

   +/ orders * prices
16

   +/ \ 1 2 3 4 5
1 3 6 10 15

   2 3 * / 1 2 3 4 5
2 4 6  8 10
3 6 9 12 15

   (cube=: ^&3) i. 9
0 1 8 27 64 125 216 343 512
```

Части речи

`50 fahrenheit`	Существительные/Именованные Существительные
`+ - * % cube`	Глаголы/Именованные Глаголы
`/ \`	Наречия
`&`	Союзы
`=:`	Присваивание
`( )`	Пунктуация

4. Таблицы Функций

В младшей школе *сложение* иллюстрируется таблицей сложения, поведение других глаголов (или *функций*) тоже можно прояснить при помощи их таблиц.

На нескольких следующих страницах показано -- как построить такие таблицы, и как использовать *служебные* функции `over` и `by` для помещения значений аргумента в рамку (чтобы результат было проще интерпретировать).

Изучите показанные ниже таблицы и постройте таблицы других функций (как например `< <.` и `%`) из Разговорника.

```
   (+/~ ; */~) 0 1 2
+-----+-----+
|0 1 2|0 0 0|
|1 2 3|0 1 2|
|2 3 4|0 2 4|
+-----+-----+
```

Таблицы сложения и умножения

```
   ^/ ~ i. 4
1 0 0  0
1 1 1  1
1 2 4  8
1 3 9 27
```

Таблица степени

```
   +./~ 0 1
0 1
1 1
```

Таблица или

5. Таблицы в Рамках

```
   over=: ({.;}.)@":@,                NB. Служебные функции. Скорее для пользо-
   by=:  ' '&;@,.@[,.]                NB. вания, чем для немедленного изучения.

   primes=: 2 3 5
   i=: 0 1 2 3 4

   primes by i over primes */ i
+-+------------+
| |0 1  2  3  4|
+-+------------+
|2|0 2  4  6  8|
|3|0 3  6  9 12|
|5|0 5 10 15 20|
+-+------------+

   tba=: 1 : '[ by ] over x/'          NB. Наречие "таблично"

   primes * tba i
+-+------------+
| |0 1  2  3  4|
+-+------------+
|2|0 2  4  6  8|
|3|0 3  6  9 12|
|5|0 5 10 15 20|
+-+------------+
   7 11 ^ tba i
+--+--------------------+
|  |0  1   2    3     4|
+--+--------------------+
| 7|1  7  49  343  2401|
|11|1 11 121 1331 14641|
+--+--------------------+
```

6. Таблицы (Частота Букв)

```
   text=: ' i sing of olaf glad and big'
   alph=: ' abcdefghijklmnopqrstuvwxyz'
   10{.alph=/text
1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

   '01'{~10{.alph=/text
10100001001000010000010001000
00000000000001000001001000000
00000000000000000000000000100
00000000000000000000000000000
00000000000000000000100010000
00000000000000000000000000000
00000000010000100000000000000
00000010000000001000000000001
00000000000000000000000000000
01001000000000000000000000010

   ]LF=: 2 13 $ +/"1 alph =/ text       NB. Таблица частоты букв
7 3 1 0 2 0 2 3 0 3 0 0 2
0 2 2 0 0 0 1 0 0 0 0 0 0

   +/+/LF
28

   $text
28
```

7. Таблицы

Таблицы делимости и остатков

```
   div=: 0=rem=: i |/i=: i.7
   (,.i) ; rem ; div
+-+-------------+-------------+
|0|0 1 2 3 4 5 6|1 0 0 0 0 0 0|
|1|0 0 0 0 0 0 0|1 1 1 1 1 1 1|
|2|0 1 0 1 0 1 0|1 0 1 0 1 0 1|
|3|0 1 2 0 1 2 0|1 0 0 1 0 0 1|
|4|0 1 2 3 0 1 2|1 0 0 0 1 0 0|
|5|0 1 2 3 4 0 1|1 0 0 0 0 1 0|
|6|0 1 2 3 4 5 0|1 0 0 0 0 0 1|
+-+-------------+-------------+
```

Простые, проверка, # делителей

```
   (i#~2=+/div);(2=+/div);(+/div)
+-----+-------------+-------------+
|2 3 5|0 0 1 1 0 1 0|7 1 2 2 3 2 4|
+-----+-------------+-------------+
```

```
   (=/~ ; </~ ; <:/~; ~:/~) i. 5
+---------+---------+---------+---------+
|1 0 0 0 0|0 1 1 1 1|1 1 1 1 1|0 1 1 1 1|
|0 1 0 0 0|0 0 1 1 1|0 1 1 1 1|1 0 1 1 1|
|0 0 1 0 0|0 0 0 1 1|0 0 1 1 1|1 1 0 1 1|
|0 0 0 1 0|0 0 0 0 1|0 0 0 1 1|1 1 1 0 1|
|0 0 0 0 1|0 0 0 0 0|0 0 0 0 1|1 1 1 1 0|
+---------+---------+---------+---------+
```

Наречие "таблично"

```
   t=: (/ ~) (@i.)
```

```
   (=t;<t;<:t;~:t) 5
+---------+---------+---------+---------+
|1 0 0 0 0|0 1 1 1 1|1 1 1 1 1|0 1 1 1 1|
|0 1 0 0 0|0 0 1 1 1|0 1 1 1 1|1 0 1 1 1|
|0 0 1 0 0|0 0 0 1 1|0 0 1 1 1|1 1 0 1 1|
|0 0 0 1 0|0 0 0 0 1|0 0 0 1 1|1 1 1 0 1|
|0 0 0 0 1|0 0 0 0 0|0 0 0 0 1|1 1 1 1 0|
+---------+---------+---------+---------+
```

```
   (^t;!t;-t) 5
+-------------+---------+-------------+
|1 0  0  0   0|1 1 1 1 1|0 _1 _2 _3 _4|
|1 1  1  1   1|0 1 2 3 4|1  0 _1 _2 _3|
|1 2  4  8  16|0 0 1 3 6|2  1  0 _1 _2|
|1 3  9 27  81|0 0 0 1 4|3  2  1  0 _1|
|1 4 16 64 256|0 0 0 0 1|4  3  2  1  0|
+-------------+---------+-------------+
```

8. Классификация

Классификация -- знакомое понятие. Например, классификация букв в алфавите по соответствующим им звукам на *гласные*, *согласные* и *шипящие*; цветов на *первичные* и *вторичные*; чисел на *четные*, *нечетные*, *простые* и *комплексные*.

Понятие классификации служит основой для многих других важных понятий, таких как графики, гистограммы и множества.

Классификация может быть *полной* (каждый обьект попадает как минимум в один класс) или *несвязной* (каждый обьект попадает максимум в один класс). *График* является *несвязной* классификацией, соответствующей связной, производящей гистограмму.

```
   x=: 1 2 3 4 5 6 7
   ]y=: (x-3) * (x-5)                    NB. Парабола (корни в 3 и 5)
8 3 0 _1 0 3 8

   range=: >./ - i.@spread
   spread=: 1: + >./ - <./
   spread y
10

   range y
8 7 6 5 4 3 2 1 0 _1

   ((range <:/ ]);{&' *'@(range<:/])) y    NB. Гистограммы y
+-------------+-------+
|1 0 0 0 0 0 1|*     *|
|1 0 0 0 0 0 1|*     *|
|1 0 0 0 0 0 1|*     *|
|1 0 0 0 0 0 1|*     *|
|1 0 0 0 0 0 1|*     *|
|1 1 0 0 0 1 1|**   **|
|1 1 0 0 0 1 1|**   **|
|1 1 0 0 0 1 1|**   **|
|1 1 1 0 1 1 1|*** ***|
|1 1 1 1 1 1 1|*******|
+-------------+-------+
```

9. Несвязная Классификация (График)

Для булевского списка b результат </b будет ненулевым тогда и только тогда, когда этот список имеет единственный ненулевой элемент, последний. Следовательно, </\ в применении к b подавляет все единицы после первой, и результат, таким образом, представляет собой несвязную классификацию. Например:

```
   cct=: #:@i.@(2: ^ #)           NB. Полная таблица классификации
   b=: |: cct 2 3 5 7
   b
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

   </b
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

   </\b
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

   y=: (x-3) * (x-5) [ x=: 1 2 3 4 5 6 7
   range=: >./ - i.@spread
   spread=: 1: + >./ - <./
   bc=: (range <:/]) y
   bc;(</\bc);({&'.*' </\bc)       NB. Гистограмма и график
+-------------+-------------+-------+
|1 0 0 0 0 0 1|1 0 0 0 0 0 1|*.....*|
|1 0 0 0 0 0 1|0 0 0 0 0 0 0|.......|
|1 0 0 0 0 0 1|0 0 0 0 0 0 0|.......|
|1 0 0 0 0 0 1|0 0 0 0 0 0 0|.......|
|1 0 0 0 0 0 1|0 0 0 0 0 0 0|.......|
|1 1 0 0 0 1 1|0 1 0 0 0 1 0|.*...*.|
|1 1 0 0 0 1 1|0 0 0 0 0 0 0|.......|
|1 1 0 0 0 1 1|0 0 0 0 0 0 0|.......|
|1 1 1 0 1 1 1|0 0 1 0 1 0 0|..*.*..|
|1 1 1 1 1 1 1|0 0 0 1 0 0 0|...*...|
+-------------+-------------+-------+
```

10. Классификация (с Выборкой и Внутренним Произведением)

Полная таблица классификации имеет множество полезных применений. В частности, интересны результаты ее внутреннего (скалярного, матричного и обобщенного) умножения на различные векторы и матрицы. Например:

```
  cct=: #:@i.@(2: ^ #)
  m=: 2 3 5 ,: 4 2 1
  n=: |: cct 0{m
  m ; n ; m +/ . * n
+-----+---------------+----------------+
|2 3 5|0 0 0 0 1 1 1 1|0 5 3 8 2 7 5 10|
|4 2 1|0 0 1 1 0 0 1 1|0 1 2 3 4 5 6  7|
|     |0 1 0 1 0 1 0 1|                |
+-----+---------------+----------------+
```

Увидеть систему в этом произведении проще всего в следующем отображении, где элемент в некоторой строке и некотором столбце произведения `p` (в правом нижнем углу) соответствует строке левого аргумента и столбцу правого. Тоесть:

```
   ('' ; n) ,: (m ; p=: m +/ . * n)
+-----+----------------+
|     |0 0 0 0 1 1 1 1 |
|     |0 0 1 1 0 0 1 1 |
|     |0 1 0 1 0 1 0 1 |
+-----+----------------+
|2 3 5|0 5 3 8 2 7 5 10|
|4 2 1|0 1 2 3 4 5 6  7|
+-----+----------------+

   (+/r*c) ; (r*c) ; (r=: 0{m) ; (c=: 3{"1 n) ; (<0 3){p
+-+-----+-----+-----+-+
|8|0 3 5|2 3 5|0 1 1|8|
+-+-----+-----+-----+-+
```

Так же как элемент обыкновенного матричного произведения представляет собой сумму произведений, элемент внутреннего произведения вида `*/ . ^` является произведением степеней. Следовательно, `m */ . ^ n` дает произведения всех возможных подмножеств строк `m`:

```
   m */ . ^ n
1 5 3 15 2 10 6 30
1 1 2  2 4  4 8  8
```

Так же см. использование для классификации наречия *по ключу* (`/.`).

11. Классификация (Множества и Утверждения)

Выражение `-.+./t` , в применении к любой таблице классификации `t` , дает таблицу полной классификации. Таким образом, функция, определенная ниже, замыкает таблицу классификации. Функция `tab` гарантирует, что скалярный или векторный аргумент будет воспринят как таблица с одной строкой.

```
   c=: complete=: (] , (+./ {. ,:)@:-.@:(+./))@:tab
   tab=: ,:^:(0:>.2:-#@$)

   c 0 0 1,:0 1 0
0 0 1
0 1 0
1 0 0

   c 1 0 1,:0 1 0
1 0 1
0 1 0

   (c 1 0 1);(c c 1 0 1);(c 0);(c 1)
+-----+-----+-+-+
|1 0 1|1 0 1|0|1|
|0 1 0|0 1 0|1| |
+-----+-----+-+-+
```

Функция, возвращающая булевский список называется *утверждением*; ее результат есть одно-направленная классификация, называемая *множеством*. Классификация, конечно, может быть замкнута добавлением дополнительного множества в качестве отдельного класса. Например:

```
   p1=: 2&<: *. <&5          Множество, определенное интервалом
   p1a=: (2:<:]) *. (]<5:)   Альтернативное определение
   p2=: = <.                 Множество целых
   a=: 2 %~ i. 11
   (],p1,p1a,p2,(p1+.p2),:(p1*.p2)) a
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
0   0 0   0 1   1 1   1 1   1 0
0   0 0   0 1   1 1   1 1   1 0
1   0 1   0 1   0 1   0 1   0 1
1   0 1   0 1   1 1   1 1   1 1
0   0 0   0 1   0 1   0 1   0 0


   list=: 1 : 'x # ]'        Наречие для перечисления элементов множес

   ((p1 list);(p2 list);((p1*.p2)list)) a
+-----------------+-----------+-----+
|2 2.5 3 3.5 4 4.5|0 1 2 3 4 5|2 3 4|
```

```
+-----------------+-----------+-----+
```

12. Сортировка

Глагол *упорядочить* `x /: y` располагает элементы `x` в порядке, необходимом для упорядочения `y` , тоесть заданном перестановкой `/:y` :

```
   x=: 2 7 1 8 [ y=: 1 7 3 2
   (/:y);((/:y){x);(x/:y);(x/:x)
+-------+-------+-------+-------+
|0 3 2 1|2 8 1 7|2 8 1 7|1 2 7 8|
+-------+-------+-------+-------+
```

Упорядочение текста основывается на порядке букв в алфавите `a.` . Например:

```
   text=:'For example, if the name "text" is used for the
present sentence (up to and including the colon), then:'    NB. одной строкой
   tdw=: >dwds=: ~. wds=: ;: text
   ($tdw),($dwds),($wds),($text)
21 9 21 25 103

   ]alph=: a. {~ ,(i. 26) +/ (a.i.'aA')
aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ

   tdw; (tdw /: tdw);(tdw/: alph i. tdw)
+---------+---------+---------+
|For      |"        |and      |
|example  |(        |colon    |
|,        |)        |example  |
|if       |,        |for      |
|the      |For      |For      |
|name     |and      |if       |
|"        |colon    |including|
|text     |example  |is       |
|is       |for      |name     |
|used     |if       |present  |
|for      |including|sentence |
|present  |is       |text     |
|sentence |name     |then:    |
|(        |present  |the      |
|up       |sentence |to       |
|to       |text     |up       |
|and      |the      |used     |
|including|then:    |,        |
|colon    |to       |"        |
|)        |up       |(        |
|then:    |used     |)        |
+---------+---------+---------+
```

Средняя колонка представляет из себя таблицу всех встреченных слов, расположенных по алфавиту. Заметьте, что, поскольку заглавные и строчные буквы не перемешаны в алфавите `a.`, слова

"for" и "For" находятся далеко друг от друга; в третьем столбце они сведены вместе путем индексирования букв подходящим алфавитом.

13. Композиции (осуществляемые союзами)

Символ ∘ служит в математике для обозначения функции, определенной как *композиция* двух функций: `f ∘ g y` эквивалентно `f (g y)` . Обычно такая композиция определена только для функций одного скалярного аргумента.

В **J** композиции можно произвести пятью различными союзами, а так же путем записи изолированных цепочек глаголов: крючков, вилок, и более длинных цепочек из них. Упомянутые пять союзов-- это `& &. &: @` и `@:` . Союзы `@` и `@:` связаны между собой подобно `&` и `&:` .

Ближе всего к математической композиции ∘ союз `&` . Он идентичен ей для случая двух скалярных (ранга ноль) функций, с последующим применением композиции к единственному скалярному аргументу. Однако, этот союз расширен в двух направлениях:

1. В применении к глаголу и существительному, он производит монадную функцию, как в случаях `10&^.` (Логарифм по основанию 10) и `^&3` (Куб).
2. В применении к двум глаголам, он производит (вдобавок к монадному случаю, используемому в математике) диадный случай, определенный как: `x f&g y` ↔ `(g x) f (g y)` . Например, `x %&! y` есть частное факториалов `x` и `y` .

Союз `&.` применим только к глаголам, `f&.g` эквивалентно `f&g` , за исключением того, что к его конечному результату применяется обратная к `g` функция. Например:

```
   3 +&^. 4                    3 +&.^. 4
2.48491                     12
```

Для скалярных аргументов функции `f&:g` и `f&g` эквивалентны, но для аргументов большего ранга `g` применяется к каждой ячейке, в соответствии со своими рангами. В случае `f&g`, функция `f` применяется потом к каждому полученному результату `g` индивидуально, а в случае `f&:g` она применяется сразу к полному результату, полученному после обработки всех ячеек. Например:

```
   [] ; %. ; |:&%. ; |:&:%.) i. 2 2 2
+---+-------+------+---------+
|0 1|_1.5 0.5|_1.5  1|_1.5 _3.5|
|2 3|   1   0| 0.5  0|   1    3|
|   |        |       |         |
```

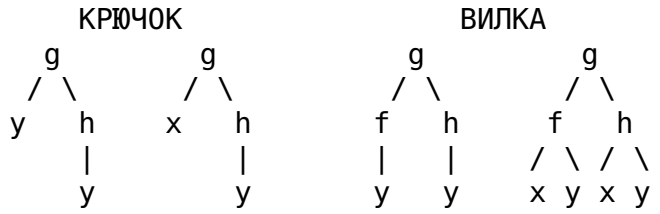
```
|4 5|_3.5 2.5|_3.5  3| 0.5  2.5|
|6 7|   3  _2| 2.5 _2|   0   _2|
+---+--------+-------+---------+
```

В монадном случае союзы `@` и `&` совпадают, как показано в таблице ниже для ячеек `x` и `y` , соответствующих рангам `g` :

```
  f&g y &harr; f g y
  f@g y &harr; f g y
x f&g y &harr; (g x) f (g y)
x f@g y &harr; f (x g y)
```

14. Композиции (с использованием Крючков и Вилок)

Цепочка двух или трех глаголов производит *глагол*, определенный диаграммами:



Например, `5(+*-)3` ↔ `(5+3)*(5-3)` .

Следующие определения взяты из Раздела II F, где используются статистические функции:

```
   mean=: +/%#                                  Среднее
   norm=: ] - +/ % #                            Центрировать на Среднем
   sqcm=: 2: ^~ ] - +/ % #                      Квадрат центрированного на сред
   var =: [:    mean 2: ^~ ] - mean=: +/%#      Квадрат Дисперсии
   stdv=: 2:%:[:mean 2: ^~ ] - mean=: +/%#      Стандартное отклонение

   ,.&.>@(];mean;norm;sqcm;var;stdv) y=: 2 3 4 5
+-+---+----+----+----+-------+
|2|3.5|_1.5|2.25|1.25|1.11803|
|3|   |_0.5|0.25|    |       |
|4|   | 0.5|0.25|    |       |
|5|   | 1.5|2.25|    |       |
+-+---+----+----+----+-------+
```

15. Сочленения

Для сочленения аргументов обычно используются четыре функции: `;` , `,` и `,.` и `,:` . Ниже, мы проиллюструем их на различных векторных и матричных аргументах:

```
   a=: 'pqr' [ b=: 'PQR'
   m=: 3 3$ 'abcdefghi' [ n=: 3 3$ 'ABCDEFGHI'

   a (; ; , ; ,. ; ,:) b
+---------+------+--+---+
|+---+---+|pqrPQR|pP|pqr|
||pqr|PQR||      |qQ|PQR|
|+---+---+|      |rR|   |
+---------+------+--+---+

   m (; ; , ; ,. ; ,:) n
+---------+---+------+---+
|+---+---+|abc|abcABC|abc|
||abc|ABC||def|defDEF|def|
||def|DEF||ghi|ghiGHI|ghi|
||ghi|GHI||ABC|      |   |
|+---+---+|DEF|      |ABC|
|         |GHI|      |DEF|
|         |   |      |GHI|
+---------+---+------+---+

   a (; ; , ; ,. ; ,:) n
+---------+---+----+---+
|+---+---+|pqr|pABC|pqr|
||pqr|ABC||ABC|qDEF|   |
||   |DEF||DEF|rGHI|   |
||   |GHI||GHI|    |   |
|+---+---+|   |    |ABC|
|         |   |    |DEF|
|         |   |    |GHI|
+---------+---+----+---+

   m (; ; , ; ,. ; ,:) b
+---------+---+----+---+
|+---+---+|abc|abcP|abc|
||abc|PQR||def|defQ|def|
||def|   ||ghi|ghiR|ghi|
||ghi|   ||PQR|    |   |
|+---+---+|   |    |PQR|
|         |   |    |   |
|         |   |    |   |
+---------+---+----+---+
```

16. Разбиения (Наречия)

В монадном применении, результаты наречий `\` , `\.` и `/.` осуществляют *префиксное*, *суффиксное* и *диагональное* разбиения. Эти наречия часто используются с арифметическими функциями, как в случаях суммирования (`+/`), перемножения (`*/`), вычисления цепной дроби (`((+%)/`). Мы проиллюстрируем их действие на глаголе упаковать (`<`), чтобы явно показать структуру разбиений:

```
   a=: 2 3 5 7 11 [ t=: 1 2 1 */ 1 3 3 1

   ,.&.>((+/\a) ; (+/\.a) ; ((+%)/\a) ; (+//.t);t)
+--+--+-------+--+-------+
| 2|28|      2| 1|1 3 3 1|
| 5|26|2.33333| 5|2 6 6 2|
|10|23| 2.3125|10|1 3 3 1|
|17|18|2.31304|10|       |
|28|11|2.31304| 5|       |
|  |  |       | 1|       |
+--+--+-------+--+-------+

   <\a
+-+---+-----+-------+----------+
|2|2 3|2 3 5|2 3 5 7|2 3 5 7 11|
+-+---+-----+-------+----------+

   <\.a
+----------+--------+------+----+--+
|2 3 5 7 11|3 5 7 11|5 7 11|7 11|11|
+----------+--------+------+----+--+

   </.t
+-+---+-----+-----+---+-+
|1|3 2|3 6 1|1 6 3|2 3|1|
+-+---+-----+-----+---+-+
```

В диадном случае, они дают классификации *в окне*, *вне окна*, и *по ключу*. Например:

```
   3 <\ a
+-----+-----+------+
|2 3 5|3 5 7|5 7 11|
+-----+-----+------+

   _3 <\ a
+-----+----+
|2 3 5|7 11|
+-----+----+

   2<\.a
+------+------+------+-----+
|5 7 11|2 7 11|2 3 11|2 3 5|
+------+------+------+-----+
```

```
   1 2 3 1 3 *//. a
14 3 55
```

17. Разбиения (союзом "в разрезе")

Левым аргументом союза *в разрезе* является функция, применяемая к разбиениям различного типа, указанного числовым правым аргументом. Мы проиллюстрируем это на примере функции *упаковать*, используя наречие `<;.` :

```
   t=: 'When eras die/their legacies/are left to/strange police/'

   ;: t
+----+----+---+-+-----+--------+-+---+----+--+-+-------+------+-+
|When|eras|die|/|their|legacies|/|are|left|to|/|strange|police|/|
+----+----+---+-+-----+--------+-+---+----+--+-+-------+------+-+

   cut=: <;.

   _2 cut ;: t
+---------------+----------------+-------------+----------------+
|+----+----+---+|+-----+--------+|+---+----+--+|+-------+------+|
||When|eras|die|||their|legacies|||are|left|to|||strange|police||
|+----+----+---+|+-----+--------+|+---+----+--+|+-------+------+|
+---------------+----------------+-------------+----------------+

   2 cut ;: t
+-----------------+------------------+---------------+------------------+
|+----+----+---+-+|+-----+--------+-+|+---+----+--+-+|+-------+------+-+|
||When|eras|die|/|||their|legacies|/|||are|left|to|/|||strange|police|/||
|+----+----+---+-+|+-----+--------+-+|+---+----+--+-+|+-------+------+-+|
+-----------------+------------------+---------------+------------------+

   (i. 3 2 2);(i. 12 12)
+-----+-----------------------------------------------+
| 0  1|  0   1   2   3   4   5   6   7   8   9  10  11|
| 2  3| 12  13  14  15  16  17  18  19  20  21  22  23|
|     | 24  25  26  27  28  29  30  31  32  33  34  35|
| 4  5| 36  37  38  39  40  41  42  43  44  45  46  47|
| 6  7| 48  49  50  51  52  53  54  55  56  57  58  59|
|     | 60  61  62  63  64  65  66  67  68  69  70  71|
| 8  9| 72  73  74  75  76  77  78  79  80  81  82  83|
|10 11| 84  85  86  87  88  89  90  91  92  93  94  95|
|     | 96  97  98  99 100 101 102 103 104 105 106 107|
|     |108 109 110 111 112 113 114 115 116 117 118 119|
|     |120 121 122 123 124 125 126 127 128 129 130 131|
|     |132 133 134 135 136 137 138 139 140 141 142 143|
+-----+-----------------------------------------------+

   (i. 3 2 2) 0 cut i. 12 12
+--------+--------------------------+-----------+
| 1  2  3| 53  54  55  56  57  58  59|105 106 107|
|13 14 15| 65  66  67  68  69  70  71|117 118 119|
|        | 77  78  79  80  81  82  83|129 130 131|
|        | 89  90  91  92  93  94  95|141 142 143|
|        |101 102 103 104 105 106 107|           |
|        |113 114 115 116 117 118 119|           |
```

+--------+----------------------------+-----------+

18. Геометрия

Применение **J** в декартовой геометрии можно проиллюстрировать, определяя функции для многоугольников в двух измерениях, вычисляющие вектора смещения между соседними вершинами, длины сторон, полу-периметр, и площадь (по формуле Герона).

Здесь мы так же дадим и более общее определение площади, не только вычисляющее ее значение со знаком (плюс, если вершины заданы в порядке против часовой стрелки), но и применимое к многогранникам в пространстве большего числа измерений (в этом случае имя *area* лучше заменить на *volume*). Это определение основано на вычислении детерминанта квадратной матрицы, полученной добавлением в конец таблицы вершин `t` строки со значениями `%!#t` . Тоесть:

```
   (length=: +/&.:*:) 5 12
13

   disp=: ] - 1&|."1
   sides=: length@disp
   semiper=: -:@(+/)@sides
   HERON=: %:@(*/)@(semiper - 0: , sides)
   area=: -/ . * @ (] , %@!@#)

   t=: 0 0 4 ,: 3 4 7
   (];(1&|."1);disp;sides;semiper;HERON;area) t
+-----+-----+-------+-----+---------+---------------+--+
|0 0 4|0 4 0| 0 _4 4|0 4 4|0.5 3.5 4|2.29129 0j0.5 0|_2|
|3 4 7|4 7 3|_1 _3 4|1 3 4|         |               |  |
+-----+-----+-------+-----+---------+---------------+--+

   tet1=:6 0 3 0,3 6 5 8,:7 4 0 5
   tet2=: 0,.=i.3 3
   tet1;(area tet1);tet2;(area tet2)
+-------+----+-------+---------+
|6 0 3 0|11.5|0 1 0 0|_0.166667|
|3 6 5 8|    |0 0 1 0|         |
|7 4 0 5|    |0 0 0 1|         |
+-------+----+-------+---------+
```

19. Символьные Функции

Для любой функции можно определить ее *символьный* вариант, который бы показывал выражение, не вычисляя его. Например:

```
   minus=: [ , '-'"_ , ]
   'a' minus 'b'
a-b

   list=: 'abcd'
   table=: 4 4$'ABCDEFGHIJKLMNOP'
   minus/list
a-b-c-d

   (minus/\list);('01'minus"0/list);(minus//.table);table
+-------+---+-------+----+
|a      |0-a|A      |ABCD|
|a-b    |0-b|B-E    |EFGH|
|a-b-c  |0-c|C-F-I  |IJKL|
|a-b-c-d|0-d|D-G-J-M|MNOP|
|       |   |H-K-N  |    |
|       |1-a|L-O    |    |
|       |1-b|P      |    |
|       |1-c|       |    |
|       |1-d|       |    |
+-------+---+-------+----+

   (,.list)=: 4 3 2 1
   (". minus/\list) ,: (-/\4 3 2 1)
4 1 3 2
4 1 3 2

   3 (minus/\ ; minus/\.) 'abcdefg'
+-----+-------+
|a-b-c|d-e-f-g|
|b-c-d|a-e-f-g|
|c-d-e|a-b-f-g|
|d-e-f|a-b-c-g|
|e-f-g|a-b-c-d|
+-----+-------+
```

20. Ориентированные Графы

Ориентированный граф есть множество *узлов* со *связями* (или *ребрами*) между некоторыми парами этих узлов. Графы можно использовать для указания приоритетов, таких, например, как порядок выполнения операций из некоторого набора (заполнение конвертов должно предшествовать запечатыванию), или *древовидной* структуры.

Связи можно задать булевской *матрицей смежности*. При необходимости, она рассчитывается из списка ребер графа.

Матрица смежности удобна для определения различных свойств графа, таких как *степень по входящим ребрам* (количество ребер, входящих в данную вершину), *степень по исходящим ребрам*, ближайшие соседи, и *замыкание* (тоесть проведение явных соединений с любым узлом, достижимым по какому-либо пути из каждого). Например:

```
   from=: 3 7 2 5 5 7 1 5 5 5 2 6 1 2 3 7 7 4 7 2 7 4
     to=: 5 6 0 2 6 2 7 6 0 7 3 3 2 1 7 0 4 2 3 0 0 3

   $ arcs=: from,.to
22 2

   |: arcs { nodes=: 'ABCDEFGH'
DHCFFHBFFFCGBCDHHEHCHE
FGACGCHGAHDDCBHAECDAAD
```

Для отображения транспонировано

```
   CM=: #. e.~ [: i. [ , [
   ]cm=: (>:>./,arcs) CM arcs
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1
1 1 0 1 0 0 0 0
0 0 0 0 0 1 0 1
0 0 1 1 0 0 0 0
1 0 1 0 0 0 1 1
0 0 0 1 0 0 0 0
1 0 1 1 1 0 1 0

   (+/cm);(+/"1 cm);  (+/+/cm);(#arcs);(#~.arcs)
+--------------+--------------+--+--+--+
|3 1 4 4 1 1 2 3|0 2 3 2 2 4 1 5|19|22|19|
+--------------+--------------+--+--+--+
```

Матрица смежности из ребер

Выше рассчитаны: степень каждого узла по входящим ребрам, степень по исходящим и полная степень всех узлов; далее следуют: полное число ребер и число *различных* ребер. Узлам можно поставить в соответствие булевский вектор `b` , тогда внутреннее произведение `b +./ . *. cm` дает (в том-же представлении) узлы,

достижимые из тех, которые были заданы в этом векторе изначально. Ближайшее окружение некоторого количества узлов (включая начальные точки) можно получить функцией `imfam` :

```
   imfam=: [ +. +./ . *.
   (b=: 1 0 0 0 0 0 0 1) imfam cm
1 0 1 1 1 0 1 1
```

21. Замыкание

Так же как `b imfam cm` дает ближайшее окружение `b`, фраза `cm imfam cm` производит ближайшее окружение каждой строки `cm`. Для наглядности возьмем теперь более разреженную матрицу смежности и будем применять к ней степени `imfam` , производя последовательные поколения ближайшего окружения. Как видим, возводя этот глагол в бесконечную степень, мы получим *замыкание* матрицы смежности. Тоесть, матрицу смежности всех точек, соединенных путями любой длины:

```
   cm=: (i. =/ <:@i.) 8
   <"2 cm imfam^:0 1 2 _ cm
+---------------+---------------+---------------+---------------+
|0 1 0 0 0 0 0 0|0 1 1 0 0 0 0 0|0 1 1 1 0 0 0 0|0 1 1 1 1 1 1 1|
|0 0 1 0 0 0 0 0|0 0 1 1 0 0 0 0|0 0 1 1 1 0 0 0|0 0 1 1 1 1 1 1|
|0 0 0 1 0 0 0 0|0 0 0 1 1 0 0 0|0 0 0 1 1 1 0 0|0 0 0 1 1 1 1 1|
|0 0 0 0 1 0 0 0|0 0 0 0 1 1 0 0|0 0 0 0 1 1 1 0|0 0 0 0 1 1 1 1|
|0 0 0 0 0 1 0 0|0 0 0 0 0 1 1 0|0 0 0 0 0 1 1 1|0 0 0 0 0 1 1 1|
|0 0 0 0 0 0 1 0|0 0 0 0 0 0 1 1|0 0 0 0 0 0 1 1|0 0 0 0 0 0 1 1|
|0 0 0 0 0 0 0 1|0 0 0 0 0 0 0 1|0 0 0 0 0 0 0 1|0 0 0 0 0 0 0 1|
|0 0 0 0 0 0 0 0|0 0 0 0 0 0 0 0|0 0 0 0 0 0 0 0|0 0 0 0 0 0 0 0|
+---------------+---------------+---------------+---------------+
```

Таким образом, замыкание `cm` можно выразить как `cm imfam^:_ cm`, , а осуществляющую его монадную функцию как:

```
   (closure=: imfam^:_ ~) cm
0 1 1 1 1 1 1 1
0 0 1 1 1 1 1 1
0 0 0 1 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 0 0 1 1 1
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
```

Фиксируя определение этого глагола, мы увидим полное определение замыкания:

```
   closure f.
([ +. +./ .*.)^:_~
```

22. Расстояние

Определим расстояние "вдоль улиц" между двумя точками как сумму модулей разности их координат вдоль каждого измерения. Тоесть:

```
   d=: +/@:|@:-"1
   p=: 3 5 1 [ q=: 7 4 0
   p d q
6

   table=: #: i. 2^3
   (]; d/~) table             NB. Таблица и расстояния между каждой парой точек
+-----+---------------+
|0 0 0|0 1 1 2 1 2 2 3|
|0 0 1|1 0 2 1 2 1 3 2|
|0 1 0|1 2 0 1 2 3 1 2|
|0 1 1|2 1 1 0 3 2 2 1|
|1 0 0|1 2 2 3 0 1 1 2|
|1 0 1|2 1 3 2 1 0 2 1|
|1 1 0|2 3 1 2 1 2 0 1|
|1 1 1|3 2 2 1 2 1 1 0|
+-----+---------------+

   g=: [ * [ = d/~@]
   (];(d/~);(1&g);(2&g)) table
+-----+---------------+---------------+---------------+
|0 0 0|0 1 1 2 1 2 2 3|0 1 1 0 1 0 0 0|0 0 0 2 0 2 2 0|
|0 0 1|1 0 2 1 2 1 3 2|1 0 0 1 0 1 0 0|0 0 2 0 2 0 0 2|
|0 1 0|1 2 0 1 2 3 1 2|1 0 0 1 0 0 1 0|0 2 0 0 2 0 0 2|
|0 1 1|2 1 1 0 3 2 2 1|0 1 1 0 0 0 0 1|2 0 0 0 0 2 2 0|
|1 0 0|1 2 2 3 0 1 1 2|1 0 0 0 0 1 1 0|0 2 2 0 0 0 0 2|
|1 0 1|2 1 3 2 1 0 2 1|0 1 0 0 1 0 0 1|2 0 0 2 0 0 2 0|
|1 1 0|2 3 1 2 1 2 0 1|0 0 1 0 1 0 0 1|2 0 0 2 0 2 0 0|
|1 1 1|3 2 2 1 2 1 1 0|0 0 0 1 0 1 1 0|0 2 2 0 2 0 0 0|
+-----+---------------+---------------+---------------+

   {&' .+*' &.> (];(d/~);(1&g);(2&g);(3&g)) table
+---+--------+--------+--------+--------+
|   | ..+.++*| .. .   |   + ++ |       *|
|  .|. +.+.*+|.  . .  |  + +  +|      * |
| . |.+ .+*.+|.  .  . | +  +  +|     *  |
| ..|+.. *++.| ..    .|+    ++ |    *   |
|.  |.++* ..+|.    .. | ++    +|   *    |
|. .|+.*+. +.| .  .  .|+  +  + |  *     |
|.. |+*.+.+ .|  . .  .|+  + +  | *      |
|...|*++.+.. |   . .. | ++ +   |*       |
+---+--------+--------+--------+--------+
```

23. Многочлены

Монадная функция `M=: 3: * ] ^ 2:` (число, кратное целой степени своего аргумента) называется *одночленом*. Сумма одночленов, как например `SM=: (3:*]^2:)+(2.5"_*]^4:)+(_5:*]^0:)` , называется *много*членом.

Любой многочлен можно выразить в *стандартном* виде `c&p`, где `c` -- список *коэффициентов*, а `p=: +/@([*]^i.@#@[)"1 0` . Например:

```
   SM=: (3:*]^2:)+(2.5"_*]^4:)+(_5:*]^0:)
   p=: +/@([*]^i.@#@[)"1 0
   c=: _5 0 3 0 2.5
   x=: _2 _1 0 1 2
   (SM x),(c p x),:(c&p x)
47 0.5 _5 0.5 47
47 0.5 _5 0.5 47
47 0.5 _5 0.5 47
```

Функция `p` эквивалентна примитиву `p.` , который мы и будем использовать в дальнейшем. Многочлены `c&p.` очень важны по ряду причин. В частности:

1. Они применимы к любому числовому аргументу: действительному или комплексному (коэффициенты `c` тоже могут быть действительными или комплексными).

2. Ими можно приблизить самые разнообразные функции.

3. Их множество *замкнуто* по отношению к ряду операций: сумма, разность, произведение, композиция `@`, производная и интеграл многочлен(а/ов) тоже являются многочленами.

4. Для перечисленных в 3 операций коэффициенты результата можно легко вычислить. Например, если `#c` равно `#d`, то `c&p. + d&p.` равняется `(c+d)&p.` . В более общем случае, `(+/c,:d)&p.` . Тоесть:

`ps=: +/@,:`	Сумма многочленов
`pd=: -/@,:`	Разность многочленов
`pp=: +//.@(*/)`	Произведение многочленов
`D=: d.1`	Скалярная (ранга 0) первая производная
`pD=: 1: }. ] * i.@#`	Производная многочлена
`pI=: 0: , ] % 1: + i.@#`	Интеграл многочлена

24. Многочлены (продолжение)

Например:

```
   c=: 1 2 1 [ d=: 1 3 3 1
   x=: 2 1 0 _1 _2
  ,.&.>((c pp d);((c pp d)&p. x);(c&p.*d&p.)x)
+--+---+---+
| 1|243|243|
| 5| 32| 32|
|10|  1|  1|
|10|  0|  0|
| 5| _1| _1|
| 1|   |   |
+--+---+---+

   ,.&.>((d&p.D x);(pD d);((pD d)&p. x);(pI d);pD pI d)
+--+-+--+----+-+
|27|3|27|   0|1|
|12|6|12|   1|3|
| 3|3| 3| 1.5|3|
| 0| | 0|   1|1|
| 3| | 3|0.25| |
+--+-+--+----+-+

   ]e=: c(ps pp pd)d
0 _2 _9 _16 _14 _6 _1

   e&p. x
_648 _48 0 0 0

   ((c&p.+d&p.)*(c&p.-d&p.)) x
_648 _48 0 0 0

   f=: c&p.(+*-) d&p.
   f x
_648 _48 0 0 0

   ]g=: pD c pp d
5 20 30 20 5

   (g&p. ,: (c&p.*d&p.) D) x
405 80 5 0 5
405 80 5 0 5
```

25. Многочлены через Корни

Произведение `*/y-r` называется *многочленом, выраженным через его корни* `r` . Это произведение можно переписать в виде многочлена от аргумента `y` , для которого `r` будет списком его *корней* или *нулей*. Например:

```
   */y-r [ y=: 7 [ r=: 2 3 5 [ x=: 7 6 5 4 3 2
40

   pp=: +//.@(*/)
   c=: pp/monomials=: (- ,. 1:) r
   cfr=: [: pp/ - ,. 1:            NB. Коэффициенты через корни
   pir=: */@(]-[)"1 0              NB. Многочлен через корни

  ,.&.>(r;monomials;c;(cfr r);(c&p. y);(r pir x))
+-+----+---+---+--+--+
|2|_2 1|_30|_30|40|40|
|3|_3 1| 31| 31|  |12|
|5|_5 1|_10|_10|  | 0|
| |    |  1|  1|  |_2|
| |    |   |   |  | 0|
| |    |   |   |  | 0|
+-+----+---+---+--+--+
```

Поскольку последний (при самой высокой степени) из возвращаемых `cfr` коэффициентов всегда равен 1, функция `pir` не позволяет задать произвольный многочлен, если только не добавить к ней дополнительный множитель. Для этого мы переопределим `cfr` и `pir` , чтобы они принимали в качестве аргумента список упакованных множителя и корней:

```
   CFR=: (* cfr)&>/
   PIR=: CFR@[ p. ]

   CFR 3;r
_90 93 _30 3

   (3;r) PIR x
120 36 0 _6 0 0
```

Далее мы продемонстрируем использование многочленов для приближения функций:

```
   ]ce=: ^ t. i. 7                 NB. Первые семь членов ряда Тейлора для экспон
1 1 0.5 0.166667 0.0416667 0.00833333 0.00138889

   (^ - ce&p.) _1 _0.5 0 0.5 1     NB. Сравнение с экспонентой
_0.000176114 _1.45834e_6 0 1.65264e_6 0.000226273
```

```
   pD ce                              NB. Экспонента равна своей производной
1 1 0.5 0.166667 0.0416667 0.00833333
```

26. Многочлены: Корни из Коэффициентов I (метод Ньютона)

Поскольку многочлены `(m;r)&PIR` и `(c=:CFR (m;r))&p.` идентичны, параметры `m;r` и `c` являются различными *представлениями* одной и той-же функции. Каждое представление имеет свои полезные свойства. Например, сложение многочленов проще выполнить, если они представлены коэффициентами, и сложнее, если корнями; нахождение нулей по заданным коэффициентам многочлена сложно, но тривиально, если многочлен ими-же и представлен. Таким образом, полезно иметь функции, выполняющие преобразование между упомянутыми представлениями. `CFR` работает в одном направлении, обратная задача решается методом последовательных приближений.

Для дифференцируемой функции `f` , разность `(f r)-(f a)` в близких точках `r` и `a` приближенно равна разности `r-a` , помноженной на наклон касательной к графику функции `f` в точке `a,f a` , тоесть производной `f` в `a` . Обратное тоже верно, разность `r-a` можно приблизить величиной `((f r)-(f a))%f D a`, а приближением к `r` будет `a+((f r)-(f a))%f D a` .

Если `f` -- многочлен `c&p.` , а `r` -- один из его корней, тогда `f r` равняется нулю. Тоесть, если `a` -- приближение к `r` , формула для `r` упрощается до `a-(f a)%f D a` . Это выражение может дать лучшее приближение к `r` и лежит в основе *метода Ньютона*, определенного ниже в виде наречия:

```
   newton=: 1 : '] - x % x D'

   f=: (c=: 12 _10 2)&p.

   f a=: 2.4
_0.48

   f newton a
1.2

   f 2
0

   f newton ^:0 1 2 3 4 _ a
2.4 1.2 1.75385 1.9594 1.99848 2

   ]a=: (^ - 4:) newton ^: 0 1 2 3 _ a=: 1
1 1.47152 1.38982 1.3863 1.38629

   ^ {: a
4
```

В частном случае применения метода Ньютона к многочленам, можно определить наречие, работающее непосредственно с коэффициентами и использующее производную многочлена `pD` вместо универсальной производной `D` :

```
   pD=: 1 }. ] * i.@#
   NEWTON=: 1 : '] - x&p. % (pD x)&p.'

   c NEWTON ^:0 1 2 3 4 _ a=: 2.4
2.4 1.2 1.75385 1.9594 1.99848 2
```

27. Многочлены: Корни из Коэффициентов II (метод Кернера)

Метод Ньютона находит корни по одному и требует хорошего начального приближения. Метод Кернера является его обобщением, позволяющим вычислить сразу *все* корни, исходя из *списка* начальных значений `a` . Его общая схема заключается в делении каждого элемента результата `f a` на производную в точке соответствующего корня. В методе Кернера предполагается, что коэффициент при старшей степени многочлена равен единице, а потому необходимо предварительно поделить все коэффициенты на последний (корни при этом не изменяются). Метод позволяет находить и комплексные корни, но только если одно из начальных значений комплексное. В качестве примера рассмотрим разложение экспоненты в ряд Тейлора (корни соответствующего многочлена комплексные):

```
   ]d=: ^ t. i.6
1 1 0.5 0.166667 0.0416667 0.00833333

   ]c=: (norm=: % {:) d
120 120 60 20 5 1

   +. a=: (init=: r.@}.@i.@#) c |a
 0.540302  0.841471 1 1 1 1 1
_0.416147  0.909297
_0.989992  0.14112
_0.653644 _0.756802
 0.283662 _0.958924

   deriv=: [: */ 0&=@{.}@(-/~ ,: 1:)

   kerner=: 1 : '] - x&p. % deriv@]'

   r=: c kerner ^:_ a
   +.(/:|) r               NB. Действ. и мнимая части корней, упоряд. по модулю
_2.18061 4.57601e_31
 _1.6495    1.69393
 _1.6495   _1.69393
0.239806    3.12834
0.239806   _3.12834

   >./|c p. r
1.04488e_13
```

Сравните эти результаты с результатами примитива `p.` для ненормированных коэффициентов `d` . Тоесть:

```
   p. 2 4 2
+-+-----+
|2|_1 _1|
```

```
+-+-----+

   ,.;}.p. d
 0.2398064j3.12834
0.2398064j_3.12834
   _1.6495j1.69393
  _1.6495j_1.69393
          _2.18061
```

Метод Ньютона тоже годится для нахождения комплексных корней:

```
   +. d NEWTON ^:0 1 2 3 _ a=: 1j1
        1        1
0.0166065  0.99639
_0.990523 0.992532
 _1.95338  1.10685
  _1.6495   1.6939
```

28. Многочлены: Лестничные

Выражение вида `*/ x + s * i. n` часто называется *факториальной функцией*, но, чтобы не путать с функцией `!` , мы будем называть ее *лестницей*. Появляющиеся в ее определении множители, меняются с шагом `s`, будто ступеньки лестницы. Лестницы полезны в страховой деятельности и в разностном анализе.

Лестница представляет собой обобщение степени `^` (совпадает с ней при `s=:0`) и получается ее *настройкой*, как в `^!.s`. Например:

```
   x + s * i. n  [ x=: 7 [ n=: 5 [ s=: _1
7 6 5 4 3

   (*/x + s * i. n);(x ^!.s n);(x ^!.0 n);(x^n)
+----+----+-----+-----+
|2520|2520|16807|16807|
+----+----+-----+-----+
```

Фраза `+/c * x^!.s i.#c` называется *лестничным* многочленом, также как `+/c*x^i.#c` называется *степенным*. Далее мы определим наречие `P` , производящее лестничный многочлен с заданной величиной шага `s` :

```
   P=: 1 : '+/@([ * ] ^!. x i.@#@[)"1 0'
   c=: 1 3 3 1 [ d=: 1 7 6 1 [ x=: 0 1 2 3 4
   (c p. x);(c 0 P x);(d _1 P x);(d p.!._1 x)
+-------------+-------------+-------------+-------------+
|1 8 27 64 125|1 8 27 64 125|1 8 27 64 125|1 8 27 64 125|
+-------------+-------------+-------------+-------------+
```

Как показано выше, лестничные многочлены эквивалентны обыкновенным с коэффициентами, выбранными подходящим образом. Более того, преобразование между этими двумя наборами коэффициентов можно осуществить при помощи матричного произведения:

```
   VM=: 1 : '[ ^!.x/ i.@#@]'
   TO=: 2 : '(x VM %. y VM)~ @i.@#'
   (0 TO _1 c) +/ . * c
1 7 6 1
```

Матрицы `0 TO _1` и `_1 TO 0` обратны друг другу, и связаны с числами *Стирлинга*:

```
   (0 TO _1 i.5);(_1 TO 0 i.5)
+---------+------------+
|1 0 0 0 0|1 0  0  0  0|
|0 1 1 1 1|0 1 _1  2 _6|
|0 0 1 3 7|0 0  1 _3 11|
```

```
|0 0 0 1 6|0 0  0  1 _6|
|0 0 0 0 1|0 0  0  0  1|
+---------+------------+
```

Лестничный многочлен можно задать и непосредственно, настроив `p.!.s` .

Словарь

J представляет собой диалект APL, формального императивного языка. Поскольку язык **J** императивный, каждое его предложение так же называется *инструкцией* и может быть *выполнено*, производя *результат*. Поскольку **J** формальный и непротиворечивый, его предложения могут быть выполнены механически при помощи компьютера. Таким образом, **J** является *языком программирования*. Имея много общего с математической записью, он так же называется аналитическим языком.

APL родился в попытке создать непротиворечивую запись для обучения и анализа в информатике. Дальнейшее развитие APL проходило под влиянием опыта его использования в различных областях, и опыта его реализации для различных вычислительных машин [1-5].

J реализован на языке C (как описано в Hui [6]) и перенесен на множество вычислительных платформ. Влияние различий между платформами минимально и сосредоточено в рамках единственного *внешнего* союза, описанного в Приложении A. См. файлы помощи для информации о других платформно-зависимых возможностях (например, доступных в системе Windows).

Введение в этой книге содержит информацию для новичков. Работы [7-9] используют **J** в различных областях математики.

I. Алфавит и Слова

Алфавит состоит из символов ASCII, тоесть *цифр*, *букв* (латинского алфавита), *подчерка* (используемого в именах и числах), (одинарной) *кавычки*, и других символов (включая пробел), называемых далее *графическими*.

Представление чисел осуществляется при помощи цифр, подчерка (одного или двух; для обозначения отрицательных чисел, бесконечности и минус бесконечности), точки (используемой при записи вещественных чисел в качестве десятичной точки, которой *необходимо* должны предшествовать одна или более цифр), буквы `e` (как в `2.4e3` для обозначения `2400` в экспоненциальной форме), и буквы `j` для отделения действительной и мнимой части комплексного числа, как в `3e4j_0.56` . Далее см. обсуждение Констант.

Числовой *список* или *вектор* представляется последовательностью чисел, разделенных пробелами. Текстовый список (вектор символов ASCII) при вводе должен быть заключен в одиночные кавычки, где пара последовательных одиночных кавычек обозначает сам символ кавычки: `'can''t'` представляет собой сокращение из пяти букв, шести-буквенного английского слова `'cannot'`. *Ac* `a:` обозначает упакованный пустой список `<$0` .

Имена (присваиваемые существительным, глаголам и другим обьектам, как в `prices=: 4.5 12`) начинаются с буквы, но могут содержать подчерки и цифры. Имя, оканчивающееся подчеком или содержащее два последовательных подчерка, является *локативом* (они обсуждаются в Разделе II.I).

Примитивы или *первичные* обозначаются графическим символом (таким как `+` для *плюс*), который может иметь одно или более *окончаний* (точка или двоеточие), как в `+.` и `+:` (обозначающих *или* и *не-или*). Примитив так же может обозначаться именем с окончанием, как в случаях `e.` и `o.` (*содержит* и *умножить на пи*).

II. Грамматика

Следующие предложения иллюстрируют шесть частей речи:

```
   fahrenheit=: 50
   (fahrenheit-32)*5%9
10

   prices=: 3 1 4 2
   orders=: 2 0 2 1
   orders * prices
6 0 8 2

   +/orders*prices
16

   +/\1 2 3 4 5
1 3 6 10 15

   bump=: 1&+
   bump prices
4 2 5 3
```

ЧАСТИ РЕЧИ

`50 fahrenheit`	Существительные
`+ - * % bump`	Глаголы
`/ \`	Наречия
`&`	Союзы
`( )`	Пунктуация
`=:`	Присваивание

Глагол действует на существительные (одно или два) и производит результат в виде существительного. Существительные, к которым применяется данный глагол, называются его *аргументами*. Глагол может иметь два различных (но обычно связанных) значения в зависимости от того -- применяется ли он к одному аргументу (справа), или к двум (слева и справа). Например, `2%5` дает `0.4` , но `%5` дает `0.2` .

Наречие действует на единственный аргумент (существительное или глагол), стоящий *слева*. Например, `+/` производит новый глагол (его можно назвать *сложить между*), суммирующий элементы списка, к которому он применяется, а `*/` дает произведение элементов списка. Союз применяется к двум аргументам (существительными или глаголам).

Пунктуация осуществляется при помощи скобок, указывающих

порядок исполнения, как в элементарной алгебре; ключевые слова (такие как `if. do. end.`) являются другим видом пунктуации, обсуждаемым в Явно (:) и Управляющие Конструкции.

Слово `=:` ведет себя как связка "есть" в русском. Например, предложение `area=: 3*4` читается как "area (площадь) есть 3 умножить на 4". Присвоенное таким образом имя `area` обозначает существительное и может играть в дальнейшем роль существительного. Подобным образом ведут себя имена, присвоенные глаголам, наречиям и союзам. Ввод одиночного имени показывает его значение. Ошибки обсуждаются в Разделе II.J (Ошибки и Прерванные Состояния).

A. Существительные

Существительные имеют три независимых характеристики: числовые они, текстовые или символы; упакованные или нет; их ранг. Все атомы любого массива должны принадлежать одному классу: числовому, текстовому, быть символами или упаковками (массив упаковок называется упаковочным). Массивы рангов 0, 1 и 2 называются так же *атом, список,* и *таблица*; или (как в математике) *скаляр, вектор,* и *матрица*. Представление чисел и текста описано в Главе I; обсуждение символов см. в определении глагола `s:` .

Массивы. Одиночный обьект, такой как `2.3` , `_2.3j5` , `'A'` или `'+'` называется атомом. Глагол, обозначаемый запятой, соединяет свои аргументы последовательно и формирует список с *размерностью* (ее можно запросить глаголом $), равной количеству присоединенных атомов. Например:

```
   $ date=: 1,7,7,6              word=: 's','a','w'
4
   |. date                       |. word
6 7 7 1                      was
```

Использованный выше глагол `|.` , называется *перевернуть*. Фраза `s$b` производит из списка `b` массив размерности `s` . Например:

```
   (3,4) $ date,1,8,6,7,1,9,1,7
1 7 7 6
1 8 6 7
1 9 1 7

   table=: 2 3$ word,'bat'
   table                         $table
saw                           2 3
bat
```

Количество атомов в размерности существительного называется его *рангом*. Элементы размерности называются *измерениями* массива, на них можно ссылаться по индексу 0, 1, 2, и т.д. Например, измерение 0 существительного `table` имеет длину 2, а измерение 1 длину 3.

Последние *k* измерений массива `b` представляют собой *ячейки ранга k* или *k-ячейки* `b`. Остальная часть вектора размерности называется остовом `b` относительно ячеек ранга *k*; если `$c` есть `2 3 4 5`, то `c` имеет остов `2 3` относительно ячеек ранга 2, остов `2 3 4 5` относительно `0`-ячеек (атомов), и пустой остов относительно 4-ячеек. Если:

```
   ] b=:2 3 4 $ 'abcdefghijklmnopqrstuvwx'
abcd
efgh
ijkl

mnop
qrst
uvwx
```

то список abcd есть 1-ячейка b, а каждая из букв в нем представляет собой 0-ячейку.

Ячейка ранга, на единицу меньшего ранга b , называется *элементом* b; атом имеет один элемент: сам атом. Например, глагол *взять* (обозначаемый {) выбирает элементы своего правого аргумента по индексу):

```
   0{b                  1{b                  0{0{b
abcd                 mnop                 abcd
efgh                 qrst
ijkl                 uvwx

   2 1{0{b              1{2{0{b              0{3
ijkl                 j                    3
efgh
```

Далее, глагол *по возрастанию* (обозначаемый /:) возвращает индексы к { , переставляющие элементы в "лексическом" порядке. Таким образом:

```
   g=: /: n=: 4 3$3 1 4 2 7 9 3 2 0
   n                    g                    g{n
3 1 4                1 0 3 2              2 7 9
2 7 9                                     3 1 4
3 2 0                                     3 1 4
3 1 4                                     3 2 0
```

Отрицательные ранги ячеек, как в случаях _2-ячейки и _1-ячейки (элемента), обозначают ячейки, соответствующие *остовы* которых имеют длину, равную абсолютной величине числа. Например, на список abcd можно сослаться либо как на _2-ячейку, либо как на 1-ячейку b .

Упаковки. Обсуждаемые до сих пор существительные называются *открытыми*, чтобы отличить их от *упаковок*, производимых глаголом *упаковать* < . Упакованные существительные отображаются в рамках, а вся упаковка целиком является атомом. Упаковка позволяет работать с любым массивом (например, списком букв, представляющим слово) как с одиночным обьектом, тоесть атомом. Таким образом:

```
   words=:(<'I'),(<'was'),(<'it')
   letters=: 'I was it'
   $words                          $letters
3                                  8

   |. words                        |. letters
+--+---+-+                        ti saw I
|it|was|I|
+--+---+-+
   2 3$words,|.words
+--+---+--+
|I |was|it|
+--+---+--+
|it|was|I |
+--+---+--+
```

B. Глаголы

Монады и Диады. Глаголы имеют два определения: одно для *монадного* случая (один аргумент), другое для *диадного* (два аргумента). Диадное определение применяется если перед глаголом стоит подходящий левый аргумент (тоесть существительное, не являющееся аргументом союза), иначе применяется монадное определение.

Монадный случай глагола называется так же *монадой*, потому мы говорим о *монаде* `%` в выражении `%4` , и о *диаде* `%` в `3%4` . Оба случая данного глагола могут (в т.ч. и одновременно) иметь пустую область определения.

Ранги Глаголов. Понятие ранга глагола тесно связано с понятием ранга существительного: глагол ранга *k* применяется к каждой k-ячейке своего аргумента. Например (используя массив `b` из <u>Раздела A</u>):

```
   ,b
abcdefghijklmnopqrstuvwx

   ,"2 b                          ,"_1 b
abcdefghijkl                   abcdefghijkl
mnopqrstuvwx                   mnopqrstuvwx
```

Поскольку глагол *разобрать* (обозначаемый `,`) применяется к своему аргументу целиком, его ранг считается *неограниченным*. Союз *с рангом* `"` , использованный в выражении `,"2` , создает производный глагол ранга 2, разбирающий каждую из 2-ячеек и дающий результат размерности 2 на 12.

Размерность результата есть остов (относительно ячеек, к которым глагол применяется) с присоединенной к нему размерностью результата применения глагола к каждой ячейке. Обычно, эти индивидуальные размерности (результатов для всех ячеек) совпадают. Но если нет, они сначала приводятся к общему рангу, добавлением единичных измерений в начало размерности результатов меньшего ранга, а затем *выравниваются* до общей размерности, добавлением соответствующих элементов *заполнителя*: пробелов для текстовых массивов, `0` для числовых, и пустых упаковок для упаковочных. Например:

```
   i."0 s=: 2 3 4                 >'I';'was';'here'
0 1 0 0                        I
0 1 2 0                        was
0 1 2 3                        here
```

Диадному случаю глагола соответствуют два ранга, относящиеся к его левому и правому аргументам. Например:

```
   p=: 'abc'
   q=: 3 5$'wake read lamp '
   p,"0 1 q
awake
bread
clamp
```

Каждый глагол имеет три внутренних ранга: монадный, левый (диадный) и правый (диадный). Определение любого глагола должно задать его поведение на ячейках внутренних рангов (обобщение на аргументы более высокого ранга производится автоматически). Фактически, ранги глагола являются верхним пределом ранга ячеек, к которым он может быть применен, но его область определения может включать и аргументы меньшего ранга. Например, Обратить Матрицу (%.) имеет монадный ранг 2, но работает с векторами, как с матрицами из одного столбца.

Соответствие. В выражении `p v q`, аргументы `v` должны *соответствовать* друг другу, в том смысле, что один остов должен быть префиксом другого, как для `p,"0 1 q` выше, и в следующих примерах:

```
   p ,"1 1 q                    3 4 5*i. 3 4
abcwake                      0  3  6  9
abcread                     16 20 24 28
abclamp                     40 45 50 55

   (i.3 4) * 3 4 5
 0  3  6  9
16 20 24 28
40 45 50 55
```

Если остов содержит `0`, глагол применяется к ячейке, составленной из атомов заполнителя. Например:

```
   ($ #"2 i. 1 0 3 4);($ 2 3 %"1 i. 0 2)
+---+---+
|1 0|0 2|
+---+---+

   ($ $"2 i. 1 0 3 4);($ 2 3 %/"1 i. 0 4)
+-----+-----+
|1 0 2|0 2 4|
+-----+-----+
```

C. Наречия и Союзы

В отличие от глаголов, наречия и союзы имеют фиксированную валентность: наречия монадны (они применяются к своему *левому* аргументу), союзы диадны.

Аргументами союзов и наречий могут служить существительные или глаголы; союз может произвести до четырех различных типов результата.

Например, u&v производит *композицию* глаголов u и v; ^&2 производит возведение *в квадрат* привязывая к функции *в степени* правый аргумент 2; 2&^ производит функцию *2-в-степени*. Соответственно, союз & можно читать по разному, в зависимости от конкретного случая; а можно читать как *с*, что примерно покрывает их все.

D. Сравнения

Если `x` или `y` -- конечное действительное или комплексное число, сравнение `x=y` производится так же, как мы определяем равенство в повседневной жизни (тоесть с разумной относительной погрешностью), результат этого сравнения равен `1` , если разность `x-y` относительно близка к нулю. Сравнение с погрешностью применяется и другими, устанавливающими отношения, глаголами, а так же глаголами *пол* и *потолок* (`<.` и `>.`); точное определение приведено в описании Равно (=). Погрешность `t` можно *настроить* союзом `!.`, как в `x =!.t y` . Глобальную погрешность можно запросить и установить при помощи `9!:18` и `9!:19` .

E. Разбор и Выполнение

Предложение обрабатывается путем выполнения его фраз в порядке, предписанном правилами разбора языка. Например, в предложении `10%3+2` , фраза `3+2` выполняется первой и производит результат, на который впоследствии делится `10`. В общем:

1. Выполнение продвигается справа налево, кроме случаев, когда встречена правая скобка. При нахождении скобки, выполняется сегмент, ограниченный ей и соответствующей левой скобкой; результат этого выполнения заменяет целый сегмент, включая ограничивающие его скобки.
2. Наречия и союзы выполняются перед глаголами; фраза `,"2-a` эквивалентна `(,"2)-a` , но не `,"(2-a)` . Более того, левым аргументом, наречия или союза считается целое предшествующее ему глагольное выражение. Таким образом, в выражении `+/ . */b` , самое правое наречие `/` применяется к глаголу, представленному фразой `+/ . *` , не к глаголу `*` .
3. Если возможно, глагол применяется диадно; тоесть, если перед ним стоит существительное, не являющееся правым аргументом союза.
4. Некоторые *цепочки* составляют глаголы и наречия, как описано в **§** F.
5. Для соответствия этих общих правил точным, приведенным ниже, может быть необходимым заключить в скобки фразы из наречий и союзов, производящие результаты, не являющиеся существительными или глаголами.

Важное следствие этих правил состоит в том, что, в выражении без скобок, правым аргументом любого глагола является результат всей фразы, стоящей справа от него. Предложение `3*p%q^|r-5` можно, таким образом, *прочитать* слева направо как: общий результат есть 3 умножить на результат остальной фразы, который есть частное от `p` и фразы после `%` , и т.д.

Разбор производится путем перемещения последовательных элементов (или их *значений* кроме случая именованных глаголов и имен, стоящих непосредственно слева от присваивания) с конца *очереди* (в нее изначально помещается маркер `&#167;`, а потом исходное предложение целиком) на вершину *стека* (содержащего в начале четыре маркера), с последующим выполнения некоторой, готовой для выполнения, части стека, заменяя ее результатом выполнения. Например, если `a=: 1 2 3`, то выражение `b=:+/2*a` будет разобрано и выполнено следующим образом (четыре маркера внизу стека не показаны):

```
&#167; b =: + / 2 * a
&#167; b =: + / 2 *           1 2 3      следующий
&#167; b =: + / 2           *  1 2 3      следующий
&#167; b =: + /          2  *  1 2 3      следующий
&#167; b =: +         /  2  *  1 2 3      следующий
&#167; b =: +            /  2 4 6    2 Диада
&#167; b =:           +  /  2 4 6      следующий
&#167; b           =:  +  /  2 4 6      следующий
&#167; b             =:  +/  2 4 6    3 Наречие
&#167; b                    =:  12    0 Монада
&#167;                   b  =:  12      следующий
&#167;                          12    7 Есть
                        &#167; 12
```

Этот пример иллюстрирует два момента: 1) Выполнение фразы 2 * 1 2 3 откладывается до тех пор, пока в стек не передается следующий элемент (наречие /); если бы этот элемент был союзом, то 2 было бы *его* аргументом, и монада * была бы применена к 1 2 3. 2) В то время как *значение* имени a перемещается в стек, имя b (поскольку оно предшествует присваиванию) помещается как есть и получает значение 12.

Разбор можно проследить при помощи трассировки в system\packages\misc\trace.ijs .

Выполнение в стеке ограничивается исключительно его *первыми четырьмя элементами*; готовность к исполнению определяется только *классом* каждого элемента (существительное, глагол, и т.д., неприсвоенное имя считается глаголом), как описано в следующей таблице разбора. Классы первых четырех элементов стека сравниваются с первыми четырьмя столбцами таблицы, при этом выбирается первая строка, для которой происходит совпадение во всех четырех столбцах. Потом, над выделенными жирным курсивом элементами этой строки производится действие, указанное в пятом столбце, и они заменяются полученным результатом. Если ни одна строка не дает совпадения, на вершину стека помещается следующий элемент из очереди.

РАЗД.	***ГЛАГОЛ***	***СУЩ.***	ЛЮБОЕ	0 Монада
РАЗД.+НГС	ГЛАГОЛ	***ГЛАГОЛ***	***СУЩ.***	1 Монада
РАЗД.+НГС	***СУЩ.***	***ГЛАГОЛ***	***СУЩ.***	2 Диада
РАЗД.+НГС	***ГЛАГОЛ+СУЩ.***	***НАРЕЧИЕ***	ЛЮБОЕ	3 Наречие
РАЗД.+НГС	***ГЛАГОЛ+СУЩ.***	***СОЮЗ***	***ГЛАГОЛ+СУЩ.***	4 Союз
РАЗД.+НГС	***ГЛАГОЛ+СУЩ.***	***ГЛАГОЛ***	***ГЛАГОЛ***	5 Вилка
РАЗД.	***СНГС***	***СНГС***	ЛЮБОЕ	6 Крючок
ИМЯ+СУЩ	***ПРИСВ.***	***СНГС***	ЛОБОЕ	7 Есть

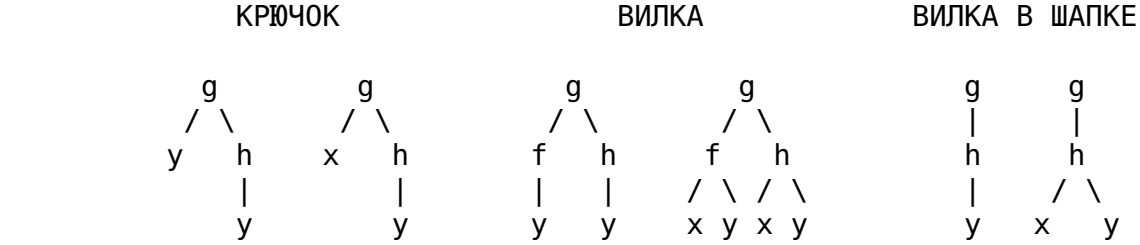
Л.СКОБКА СНГС ***П.СКОБКА*** ЛЮБОЕ 8 Скобки

Где: НГС означает НАРЕЧИЕ+ГЛАГОЛ+СУЩ.
СНГС означает СОЮЗ+НАРЕЧИЕ+ГЛАГОЛ+СУЩ.
РАЗД. означает МАРКЕР+ПРИСВ.+Л.СКОБКА

F. Цепочки

Изолированная последовательность, такая как `(+ */)` , которую "обычные" правила разбора не позволяют классифицировать как некоторую часть речи, называется *цепочкой* и может быть разобрана как описано ниже.

Цепочка двух или трех глаголов производит глагол, и (по индукции) цепочка глаголов любой длины производит глагол. Например, цепочки `+-*%` и `+-*%^` эквивалентны `+(-*%)` и `+-(*%^)`. Происходит это в соответствии со следующими правилами:



Например, `5(+*-)3` есть `(5+3)*(5-3)`. Если `f` --шапка (`[:`) то соответствующая ветвь отключается и вилка упрощается до `g h y` и `g x h y` . Цепочка `N g h` (существительное и два глагола) эквивалентна `N"_ g h` . Крючок и вилка имеют бесконечные ранги.

Двух-элементная цепочка союза и существительного или глагола производит наречие. Например, `&.>` производит наречие, которое можно назвать "each" ("в каждом"), а наречие `bc=:<"` можно назвать "box cells" ("упаковать ячейки"), поскольку, например, `0 bc x` упакует атомы `x` .

Наконец, цепочка двух наречий производит наречие, и (по индукции) цепочка любого количества наречий производит наречие. Например, `/\` есть наречие "между префиксно", и `~/~` есть "commuted table" ("перевернутая таблица"). Например:

```
   is=:/\
   + is 1 2 3 4 5
1 3 6 10 15

   ct=: ~/~
   - ct 1 2 3
 0  1 2
_1  0 1
_2 _1 0
```

G. Расширенная и Рациональная Арифметика

Целые константы произвольной точности можно ввести как последовательность цифр, оканчивающуюся `x`. В применении к обычным целым, монада `x:` производит целые произвольной точности (расширенные целые). Например, 2-х элементный вектор `1234x 56x` (или `1234 56x`) эквивалентен `x: 1234 56`. Различные примитивы, в применении к аргументам произвольной точности, дают результаты произвольной точности. Некоторые глаголы `f` дают (приближенный) действительный результат для некоторых значений аргумента из за того, что результат не является целым числом; однако, `<.@f` и `>.@f` , в применении к аргументам произвольной точности, всегда производят результаты произвольной точности. Сравнения между расширенными целыми выполняются точно. Например:

```
   !40
8.15915e47

   !40x
815915283247897734345611269596115894272000000000

   */ x: >: i.40
815915283247897734345611269596115894272000000000

   0j_25 ": ! 2000x * i. 5 1            NB. Экспоненциальная запись, 25 цифр
  1.0000000000000000000000000e0
  3.3162750924506332411753934e5735
  1.8288019515140650133147432e12673
  2.6839997657267395961163166e20065
  5.1841810604808769398058198e27752

   ] r=: <.@%: 2 * 10 ^ 56x
141421356237309504880168872420

   ,. *: r + _1 0 1
199999999999999999999999999996897269790410013239490859208 1
199999999999999999999999999999725696915156203337094236656 4
200000000000000000000000000002554124039902393434697614104 9
```

Рациональные числа можно ввести как десятичные цифры числителя и знаменателя, разделенные `r` ; в начале такой константы может стоять знак. Тоесть `3r4` есть рациональное число три четверти, а `_12r5` есть минус двенадцать пятых. Рациональные числа хрянятся и отображаются в сокращенном виде, в котором числитель и знаменатель не имеют общих множителей, а знаменатель положителен. Тоесть:

```
   1r2 _1r2 2r4 2r_4 _2r_4 0r9 5 _5
1r2 _1r2 1r2 _1r2 1r2 0 5 _5
```

Если аргументы являются рациональными, различные примитивные глаголы производят (точные) рациональные результаты; иррациональные глаголы производят (приближенные) действительные или комплексные результаты в применении к рациональным аргументам если *не все* рациональные аргументы глагола приводят к рациональному результату. (Например, `%:y` дает рациональные результаты, если все атомы `y` являются точными квадратами; `^0r1` есть действительное число.) Частное двух целых произвольной точности есть целое произвольной точности (если они делятся друг на друга) или рациональное число (если нет). Сравнения между рациональными числами выполняются точно. Диадные глаголы (например `+ - * % , = <`), требующие преобразования типов аргументов, выполняют его в соответствии со следующей таблицей:

```
    |  B  I  X  Q  D  Z
  ---+------------------
  B  |  B  I  X  Q  D  Z     B - булевский
  I  |  I  I  X  Q  D  Z     I - целый
  X  |  X  X  X  Q  D  Z     X - целый произв. точн.
  Q  |  Q  Q  Q  Q  D  Z     Q - рациональный
  D  |  D  D  D  D  D  Z     D - действительный
  Z  |  Z  Z  Z  Z  Z  Z     Z - комплексный
```

Например, в выражении `2.5+1r2` перед сложением `1r2` преобразуется в `0.5`, а результат -- действительное число `3`. А в выражении `2+1r2`, перед сложением `2` преобразуется в `2r1` , и результат -- `5r2`.

В частности, это означает, что сравнения между действительными и рациональными числами выполняются с учетом погрешности, поскольку перед сравнением рациональное число преобразуется в действительное.

Глагол `x:` производит рациональное приближение действительных аргументов.

```
   2%3
0.666667

   2%3x
2r3

   (+%)/\10$1                          NB. Действ. приближ-я к золотому сечению
1 2 1.5 1.66667 1.6 1.625 1.61538 1.61905 1.61765 1.61818
```

```
   (+%)/\x: 10$1                          NB. Рациональные версии того-же
1 2 3r2 5r3 8r5 13r8 21r13 34r21 55r34 89r55

   |: 2 x: (+%)/\x: 10$1
1 2 3 5 8 13 21 34 55 89
1 1 2 3 5  8 13 21 34 55

   (+%)/ 100$1r1
57314784401381708410 1r3542248481792619150 75

   0j30 ": (+%)/100$1r1                   NB. Показать 30 цифр после точки
1.618033988749894848204586834366

   H=: % @: >: @: (+/~) @: i. @ x:        NB. Матрица Гильберта порядка n

   ] h=: H 6
  1 1r2 1r3 1r4  1r5  1r6
1r2 1r3 1r4 1r5  1r6  1r7
1r3 1r4 1r5 1r6  1r7  1r8
1r4 1r5 1r6 1r7  1r8  1r9
1r5 1r6 1r7 1r8  1r9 1r10
1r6 1r7 1r8 1r9 1r10 1r11

   -/ .* h                                NB. Определитель h
1r186313420339200000

   ~. q: % -/ .* h                        NB. Разл. простые множ. обр. определителя
2 3 5 7 11

   i.&.(p:^:_1) 2*#h                      NB. Простые, меньшие 2*n
2 3 5 7 11

   ^ 2r1                                  NB. ^y действительное или комплексное
7.38906

   %: 49r25                               NB. %: от рационального квадрата рациональ
7r5

   %: 49r25 10r9
1.4 1.05409

   %: _2r1
0j1.41421

   1 = 1+10r1^_15                         NB. Точное (рациональное) сравнение
0

   (1.5-0.5) = 1+10r1^_15                 NB. Приближенное (действительное) сравнен
1

   0.5 = 1r2
1
```

H. Разделители и Сценарии

Операционные системы обычно используют символы "перевод строки" и/или "возврат каретки" (`10{a.` и/или `13{a.`) в качестве *разделителей* между отдельными *строками*. Текстовый список, содержащий ноль или более разделителей называется *сценарием*.

Как описано в Приложении A, сценарий `t` может быть записан и прочитан выражениями вида `t 1!:2 <'abc'` и `t=: 1!:1 <'abc'`, он так же может быть *выполнен* выражением `0!:11 <'abc'`.

Удобный ввод сценариев можно осуществить фразой `0 : 0`. Последующие нажатия клавиш считываются как текст, а клавиша *Ввод* (которая обычно оканчивает ввод выражения) воспринимается как разделитель. Ввод сценария заканчивается вводом одиночной правой скобки, добавляющей в конец списка последний разделитель. Например:

```
   s=: 0 : 0
y*%:y
:
x*!y
)

   a. i. s                    Символ с индексом 10 отмечает конец каждрй
121 46 42 37 58 121 46 10 58 10 120 46 42 33 121 46 10
```

Упаковочное и табличное представления сценария `s` можно получить следующим образом:

```
   ]b=: <;._2 s               В разрезе с разд. в конце, исключая разделите
+-----+-+----+
|y*%:y|:|x*!y|
+-----+-+----+

   ]t=: >b
y*%:y
:
x*!y
```

Любое из этих представлений `r` можно использовать как правый аргумент союза *явно* для производства наречия (`1 : r`), союза (`2 : r`), или глагола (`3 : r` или `4 : r`). Например:

```
   f=: 3 : s
   f 9                        Двоеточие в сценарии отделяет монаду и диад
27

   3 f 4                       x и y обозначают левый и правый аргументы
```

Выражения вида `a=: 1 : 0`, `c=: 2 : 0` и `v=: 3 : 0` позволяют вводить наречия, союзы и глаголы непосредственно с клавиатуры.

Файлы сценариев могут определять функции и другие обьекты, расширяющие набор первычных примитивов языка **J**. В соответствии с их генеалогией, они обычно называются *вторичными* или *третичными*.

I. Локативы

Локатив abc_f_ ссылается на имя abc в поле имен f; косвенный локатив abc__xy ссылается на abc в поле имен, название которого является текущим значением xy. Для совместимости с предыдущими версиями принимается и нестандартный локатив abc__ , эквивалентный abc_base_. Тоесть:

```
   b=: 1
   Rome=: 2
   Rome_NewYork_=: 20
   f_NewYork_=: 3 : '3*b=: Rome+y'
   f_NewYork_ 10
90

   b,Rome
1 2

   b_NewYork_
30
```

Имя является глобальным, если оно не присвоено =. внутри явного определения (:). Любое глобальное имя исполняется в текущем поле имен. В начале, текущим является поле имен base. Локатив f_abc_, во время своего исполнения переключает текущее поле имен на abc. Глагол 18!:4 так же переключает текущее поле имен, а 18!:5 возвращает его название.

Имя f_abc_ *выполняется в* поле имен abc в том смысле, что любое глобальное имя, на которое происходит ссылка в f , ищется в abc, и, если не найдено, в пути поиска поля имен abc (но выполняется все равно в abc). Путь поиска поля имен, изначально установлен в ,<,'z' , кроме поля z , изначально имеющего пустой путь. Его можно изменить при помощи 18!:2 .

Имена в полях имен обычно создаются сценарием, при помощи соответствующего выбора имени глагола, используемого для его выполнения. Например, если файл stats содержит сценарий:

```
   mean=: sum % #
   sum=: +/
```

Тогда:

```
   ssx_z_=: 0!:10              Выполнение сценария без вывода

   ssx_a_ <'stats'             Заполнить поле имен a

   mean=: 'in base locale'
```

```
   mean_a_ 3 4 5
4

   ssx_bc_ <'stats'          Заполнить поле имен bc
   sum_bc_ 3 4 5
12
```

Этот пример так же иллюстрирует использование *путей поиска имен* (состоящего из поля имен z): Сначала, утилита ssx определяется в z . При выполнении ssx_a_, ssx не обнаруживается в поле имен a и, соответственно, ищется (и находится) в поле имен z. Поскольку ssx_a_ *выполняется* в поле имен a, имена из сценария stats *определяются* в поле имен a, наполняя его. Подобное происходит и для ssx_bc_.

См. 18!: в Приложении A, а так же лабораторные "Locales" и "Object Oriented Programming", распространяемые с системой.

J. Ошибки и Прерванные Состояния

При возникновении ошибки, выполнение предложения *прерывается* с выводом сообщения об ошибке и информации о контексте, в котором она возникла. Место остановки разбора помечается четырьмя пробелами. Как иллюстрируют следующие примеры, прерывание может возникнуть при непосредственном исполнении, исполнении файла сценария, исполнении пользовательских глагола, наречия или союза:

Непосредственное исполнение

```
   2+'a'
|domain error
|   2    +'a'
```

Исполнение файла сценария

```
   t=: '2*3',(10{a.),'2+''a''',(10{a.),'2+3'
   t                          Сценарий
2*3
2+'a'
2+3

   t 1!:2 <'test'             Записать в файл

   0!:011 <'test'             Выполнить файл, прод. при ошибке, показывать
2*3
6

   2+'a'
   2+3
5

   0!:001 <'test'             Выполнить файл, ост. при ошибке, показывать (
   2*3
6

   2+'a'
|domain error
|   2    +'a'
|[-2]
```

Пользовательский глагол

```
   g=: 3 : ('1+y' ;  ':' ; '2+x+y')

   3+g 'a'
|domain error: g
|   1    +y
```

```
   13!:0 (1)                Разрешить прерывание
   3+g 'a'
|domain error: g
|   1    +y
|g[0]

      y                     Отступ в шесть пробелов говорит о прерванном
a

      y=. 12                Изменить локальное значение y

      13!:4 ''              Продолжить выполнение с текущей строки
16                          Результат, с учетом измененного y
```

В прерванном состоянии можно выполнять предложения, просматривать и изменять значения локальных имен и продолжить исполнение. Ошибки вызывают прерывание *только* если это явно разрешено (фразой `13!:0]1`). В прерванном состоянии отступ для ввода состоит из шести пробелов.

Прерывания и механизмы отладки управляются глаголами семейства `13!:` , как описано в Приложении A. Полный список ошибок приведен в Приложении D.

III. Определения

Каждое описание в Разговорнике (основной части словаря) имеет заголовок с неформальными именами монадного (слева) и диадного (справа) случаев данной функции. Кроме того, заголовок содержит формальное имя функции, состоящее из графического символа с нулем или более окончаний (точкой или двоеточием). Перед формальным именем каждого союза стоит `m` или `u` (обозначающие аргумент в виде существительного или глагола), а после него `n` или `v`. Наречия не имеют аргументов справа от имени.

В заголовке приводятся три ранга (*монадный*, *левый*, и *правый*), символом `_` обозначается бесконечный (неограниченный) ранг. Ранги могут явно зависеть от рангов аргумента, обозначенных `mu` (монадный ранг левого глагола), `lv`, и т.д.

Некоторые из примеров могут использовать функции, описанные в другой части словаря. Разбирая такие примеры, стоит игнорировать все, кроме относящихся непосредственно к данному примеру, аспекты поведения неизвестных функций, проверяя (например, путем исполнения) частичные фразы, используемые для построения окончательного результата.

Например, в описании наречия `/` , перепечатанные ниже предложения иллюстрируют варианты его использования в удобном для сравнения виде:

```
   x=: 1 2 3 4 5 [ y=: 7 5 3
   (,.x);(x+/y);y;(x*/y);(+/y);(*/y)
+-+-------+-----+--------+--+---+
|1| 8  6 4|7 5 3| 7  5  3|15|105|
|2| 9  7 5|     |14 10  6|  |   |
|3|10  8 6|     |21 15  9|  |   |
|4|11  9 7|     |28 20 12|  |   |
|5|12 10 8|     |35 25 15|  |   |
+-+-------+-----+--------+--+---+
```

Даже если внешние функции `;` и `,.` незнакомы, их действие, возможно, очевидно; Если нет, его можно проверить путем следующих экспериментов:

```
   x;y                      ,.7 8             $,.7 8
+---------+-----+          7                 2 1
|1 2 3 4 5|7 5 3|          8
+---------+-----+
```

Несмотря на то, что для каждого слова в словаре дается неформальное имя (как *внешний* для `!:`), можно использовать и

другие имена в дополнение к указанным (или вместо них). Например, `!` можно назвать *ё*, поскольку восклицательный знак происходит от I, помещенного поверх o, в сокращении латинского *io*. Подобно, вместо *целые* и *индекс в* для `i.` можно говорить *йота*.

Разговорник (Константы Конструкции Внешние Части Речи)

= Найти в Себе • Равно
< Упаковать • Меньше
> Распаковать • Больше
_ Знак "минус" / *Бесконечность*

+ Сопряжение • Прибавить
* Выделить Знак • Умножить
- Поменять Знак • Вычесть
% Обратить • Разделить

^ Экспонента • В Степени
$ Взять • Придать (Размерность)
~ *Окружая* • ***Зеркально*** / ***Вызвать***
| Модуль • Остаток

. По Минорам &#149 **Скалярно**
: Явно / **Монада-Диада**
, Разобрать• Присоединить
; Поломать • Упаковать и Наклеить

Сосчитать • Копировать
! Факториал • Кол-во Сочетаний
/ Между • ***Таблично***
\ Префиксно • ***В Окне***

[(Пропустить) • Левый
] (Пропустить) • Правый
{ Прямое Произведение • Выбрать
} *Выбрав • Заменяя* (m} u})

" С Рангом (m"n u"n m"v u"v)
` И/Или (Герундий)
@ Поверх
& С / За
? Бросить (кубик) • Сдать (карты)

=. Есть (Здесь)
<. Пол • Меньшее (Min)
>. Потолок • Большее (M
_. *Неопределенность*

+. Действ. / Мнимая • H
*. Модуль / Аргумент •
-. Не • Исключить
%. Обратить • Разделить

^. Натуральный • (Лога
$. Проредить
~. Построить Множество &#14
|. Перевернуть • Сдвин

.. Четно
:. Обратный
,. Разобрать Элементы &#149
;. В Разрезе

#. (Из) Двоичной • N-ри
!. Настроить
/. Диагонально • ***По***
\. Суффиксно • ***Вне О***

{. Голова • Взять
}. Обезглавить • Отброс

". Выполнить • В Числа

@. Сообразно
&. &.: Преобразуя (Дуально
?. Бросить • Сдать (пов

a. *Алфавит*
b. ***Поразрядно /*** Свойства
D. **Производная**
E. • Найти

i. Целые • Индекс B
j. Мнимое • Комплексное
M. ***Запоминая***
p. Корни • Многочлен (M.)

q: (П.) Множители • Степени
S: **Извлекая Уровень**
T. **Приближение Тейлора**
_9: to 9: Постоянные Функции

a: *Ac* (Пустая Упаковка)
C. Циклы • Разместить
D: **Секущая (наклон)**
f. ***Фиксируя***

i: Лестница • Индекс c
L. Сосчитать Уровни •
NB. Комментарий
p.. Производная • Инте

r. Направление • Поляр
t. ***Коэфф. Тейлора*** (m t. u t
u: Уникод (Unicode)

Приложение E. Части Речи

Глаголы

= Найти в Себе • Равно

< Упаковать • Меньше

<. Пол • Меньшее (Min)

<: Минус 1 • Меньше или Равно

> Распаковать • Больше

>. Потолок • Большее (Max)

>: Плюс 1 • Больше или Равно

_: Бесконечность

+ Сопряжение • Прибавить

+. Действ. / Мнимое • НОД (Или)

+: Удвоить &#149 Не-Или

* Выделить Знак • Умножить

*. Модуль / Аргумент • НОК (И)

*: В Квадрате • Не-И

- Поменять Знак • Вычесть

-. Не • Исключить

-: Пополам • Совпадает

Наречия

~ ***Окружая*** • ***Зеркально*** / ***Вызвать***

/ ***Между*** • ***Таблично***

/. ***Диагонально*** • ***По Ключу***

\ ***Префиксно*** • ***В Окне***

\. ***Суффиксно*** • ***Вне Окна***

} ***Выбрав • Заменяя*** (m} u})

b. ***Поразрядно /*** Свойства

f. ***Фиксируя***

M. ***Запоминая***

t. ***Коэфф. Тейлора*** (m t. u t.)

t: ***Коэфф. Тейлора (взвеш.)***

Другие

=. Есть (Здесь)

=: Есть (Всюду)

_ Знак "минус" / *Бесконечность*

_. *Неопределенность*

a. *Алфавит*

a: *Ac (Пустая Упаковка)*

NB. Комментарий

Союзы

^: В Степени (u^:n u^:v)

. По Минорам &#149 Скалярно

.. Четно

.: Нечетно

: Явно / Монада-Диада

:. Обратный

:: Ошибочный

;. В Разрезе

!. Настроить

!: Внешний

" С Рангом (m"n u"n m"v u"v)

` И/Или (Герундий)

`: Выполнив (Герундий)

@ Поверх

@. Сообразно

@: Над

& С / За

&. Преобразуя (Дуально)

&.: Преобразуя (Дуально)

&: После

d. Производная

D. Производная

D: Секущая (наклон)

H. Гипергеометрическая ф-я

L: На Уровне

S: Извлекая Уровень

T. Приближение Тейлора

% Обратить • Разделить

%. Обратить • Разделить (Матр.)

%: Квадратный Корень • Корень

^ Экспонента • В Степени

^. Натуральный • (Логарифм)

$ Взять • Придать (Размерность)

$. Проредить

$: Вызвать Себя

~. Построить Множество •

~: Найти Множество • Не-Равно

| Модуль • Остаток

|. Перевернуть • Сдвинуть

|: Транспонировать

, Разобрать • Присоединить

,. Разобрать Элементы • Сшить

,: Собрать • Наклеить

; Поломать • Упаковать и Наклеить

;: Упаковать Слова • К.

Автомат
Сосчитать • Копировать
#. (Из) Двоичной • N-ричной
#: (В) Двоичную • N-ричную
! Факториал • Кол-во Сочетаний
/: По Возрастанию • Упорядочить
\: По Убыванию • Упорядочить
[(Пропустить) • Левый
[: Сар
] (Пропустить) • Правый
{ Прямое Произведение • Выбрать
{. Голова • Взять
{: Хвост •
{:: Пути • Достать
}. Обезглавить • Отбросить
}: Купировать •
". Выполнить • В Числа
": В Текст • Форматировать
? Бросить (кубик) • Сдать (карты)

?. Бросить • Сдать (повторяемо)

A. Индекс Перест. • Переставить

C. Циклы • Разместить

e. Найти в Обломках • Содержит

E. • Найти

i. Целые • Индекс B

i: Лестница • Индекс с Конца B

I. Индексы • Индекс Интервала

j. Мнимое • Комплексное

L. Сосчитать Уровни •

o. Умножить на Пи • Круговая ф-я

p. Корни • Многочлен (M.)

p.. Производная • Интеграл (M.)

p: Простые

q: (Простые) Множители • Степени

r. Направление • Полярные

s: Символ

u: Уникод (Unicode)

x: Неограниченная

Точность
_9: to 9:
Постоянные
Функции

Найти в Себе = _ 0 0 Равно

=y находит члены множества элементов y (тоесть ~.y) в массиве y , производя булевскую таблицу размерности #~.y на #y . Например:

```
   y=: 3 3 $ 'abcdef'
   y ; (~.y) ; (=y)
+---+---+-----+
|abc|abc|1 0 1|
|def|def|0 1 0|
|abc|   |     |
+---+---+-----+
```

x=y дает 1 при x равном y , иначе 0 .

Если один из аргументов (x или y) является конечным действительным (или комплексным) числом, сравнение производится с *погрешностью* t, равной по умолчанию 2 в степени _44 и настраиваемой союзом !. (как в x=!.0 y). Формально, x=y дает 1, если модуль x-y не превышает величины t, помноженной на больший из модулей x и y .

Сравнение с погрешностью используется и другими глаголами. В частности, Совпадает (-:), Пол (<.), и Выделить Знак (*), но не глаголом Упорядочить (/:).

Глагол = (монада и диада) применим к существительным любого ранга, упаковочным и простым. Например:

```
   ]a=: ;: 'Try and try and try again.'
+---+---+---+---+---+------+
|Try|and|try|and|try|again.|
+---+---+---+---+---+------+

   ~. a
+---+---+---+------+
|Try|and|try|again.|
+---+---+---+------+

   =a
1 0 0 0 0 0
0 1 0 1 0 0
0 0 1 0 1 0
0 0 0 0 0 1

   a = <'and'
0 1 0 1 0 0
```

Поскольку точность компьютера ограничена, математически эквивалентные выражения (такие как `144*(13%144)` и `13`) могут приводить к разным результатам; использование погрешости при сравнении позволяет установить соответствие (результат `1` при сравнении) и в этом случае. Более либо менее точные сравнения можно делать с использованием союза `!.` для установки погрешности `t`, как в функции `eq=: =!.t`.

Есть =. Здесь =: Всюду

Позволяет присвоить имя обьекту, как в `a=:3` и `sum=:+/ .` В случае **=.** имя определяется *локально* как обсуждается в Явно (:), а в случае **=:** *глобально* (отметим, что глобальное присваивание локальных имен не допускается, а присваивание локативов всегда глобально).

Имена (одно или несколько) можно задавать и *косвенно*, в виде текстового (или упаковочного) списка. Если `y` -- выражение справа от присваивания, то заданное единственное имя присваивается `y` , иначе, если задано несколько имен, то каждому присваивается `>y` , если `y` атом, либо `i`-е имя присваивается `>i{y` , если `y` массив. Если текстовый список начинается символом `` ` `` , имена назначаются элементам герундия, стоящего справа.

Например:

```
   f=: 3 : 0
a=. +:y
b=: *:a
10*b
)

   a=: b=: 678
   a,b
678 678

   f 3
360

   a,b
678 36

   x=: 'abc';'c'
   (x) =: 3 4 ; 5 6 7
   abc
3 4
   c
5 6 7
```

Земетьте, что скобки вокруг имени `x` приводят к его вычислению до выполнения присваивания.

```
   'alpha beta'=: i.2 4
   alpha
0 1 2 3

   beta
```

```
4 5 6 7

   ``sum sqrt'=: +/ ` %:
   sum 3 1 4 2
10
   sqrt 2
1.41421
```

Упаковать < _ 0 0 Меньше

<y есть *атомная упаковка* y , как обсуждается в Разделе II A. Результат имеет ранг 0, и может быть *распакован* глаголом > .	x<y равно 1 при x меньшем y, с учетом погрешности. О сравнении с погрешностью см. Равно (=). <!.t использует погрешность t .

Упаковка производится и другими глаголами, такими как Упаковать и Наклеить (;) и Упаковать Слова (;:):

```
   (<'abc'),(<5 7),(<i.2 3)
+---+---+-----+
|abc|5 7|0 1 2|
|   |   |3 4 5|
+---+---+-----+

   ;: 'Now is the time'
+---+--+---+----+
|Now|is|the|time|
+---+--+---+----+

   ] a=: 2;3 5;7 11 13
+-+---+-------+
|2|3 5|7 11 13|
+-+---+-------+

   >a
2  0  0
3  5  0
7 11 13
```

Применение < в разрезе (;.) имеет несколько полезных приложений (в зависимости от правого аргумента); фраза <@v избегает выравнивания (приведения к общей размерности, и связанных с ним ошибок), необходимого в некоторых случаях применения одиночного v :

```
   <;._1 '/i sing/of olaf/'
+------+-------++
|i sing|of olaf||
+------+-------++

   i."(0) 2 3 4
0 1 0 0
0 1 2 0
0 1 2 3

   <@i."(0) 2 3 4
```

```
+---+-----+-------+
|0 1|0 1 2|0 1 2 3|
+---+-----+-------+
```

Если y имеет большой ранг, использование глаголов <"_1 y или <"_2 y часто приводит к его отображению в более удобном виде, чем просто вызов y . Отображение упакованного массива может быть нарушено присутствием в нем управляющих символов (таких как "перевод каретки" и "возврат строки"), при отображении они отображаются пробелами. Например, попробуйте выполнить < 8 32 $ a.

Пол <. 0 0 0 Меньшее (Min)

<. y дает *пол* для величины y, тоесть наибольшее целое, меньшее либо равное y . Таким образом:

```
   <. 4.6 4 _4 _4.6
4 4 _4 _5
```

Подразумеваемое при этом сравнение выполняется с погрешностью, как описано в Равно (=), которую можно настроить при помощи <.!.t . Случай комплексных аргументов описан ниже.

x<. y есть меньшее из x и y . Например:

```
   3 <. 4 _4
3 _4

   <./7 8 5 9 2
2

   <./\7 8 5 9 2
7 7 5 5 2
```

Для комплексных аргументов, определение <. моделируется глаголом:

```
floor=: j./@(ip+(c2>c1),c1+:c2)
'`c1 c2 fp ip'=:(1:>+/@fp)`(>:/@fp)`(+.-ip)`(<.@+.)
```

Эта функция, разработанная в McDonnell [10], имеет следующие свойства:

Транзитивность:	Если (<.z1)=(<.z2) и z3 принадлежит отрезку z1 - z2, то (<.z3)=(<.z1) .
Инвариантность:	(трансляционная) Если z4 есть гауссово целое, тогда (z4+<.z5)=(<.z4+z5) .
Совместимость:	(<.x j.0)=((<.x)j.0) и (<.0 j.x)=(0 j.(<.x))

Функцию <. можно рассматривать как покрытие комплексной плоскости прямоугольниками единичной площади. Все числа, попадающие в данный прямоугольник, имеют одинаковый пол. Первый прямоугольник имеет вершины в 1j0 и 0j1, а его противоположная сторона проходит через начало координат. Прямоугольники вдоль последовательных диагоналей смещаются на половинную (либо удвоенную, смотря какое семейство диагоналей выбрано за основу) их длину в направлении перпендикулярных

диагоналей.

Фраза `j./@ip` вычисляет "пол" действительной и мнимой частей комплексного аргумента по отдельности. Более того, пол `<.y` равен `->.-y` . Другими словами, он является двойственным *потолку* по отношению к (тоесть *под преобразованием*) перемене знака: `<.` ↔ `>.&.-` и `>.` ↔ `<.&.-` . Таким образом:

```
   (>.&.- ; <.) 4.6 4 _4 _4.6
+---------+---------+
|4 4 _4 _5|4 4 _4 _5|
+---------+---------+
```

Выражение `<.x+0.5` дает целое, *ближайшее* к действительному аргументу `x` . Число цифр, необходимое для представления положительного целого, есть единица плюс *пол* его логарифма по основанию десять:

```
   a ,. (,. 1:+<.) 10^. a=: 9 10 11 99 100 101
  9 0.954243 1
 10        1 2
 11  1.04139 2
 99  1.99564 2
100        2 3
101  2.00432 3
```

Минус 1 `<:` `0 0 0` Меньше или Равно

`<:y` есть `y-1` . Например:

```
   <: 2 3 5 7
1 2 4 6
```

Также см. Не (`-.`) .

`x<:y` дает `1` при `x` меньшем или равном `y` , иначе `0` . См. Равно (`=`) для описания сравнений с погрешностью. Погрешность `<:` можно так же настроить союзом `!.` .

Обратный к монадному `<:` глагол есть `>:` (Плюс 1). Например:

```
   n=: 5

   <: ^: _1 n
6

   <:^: 0 1 2 n              NB. Здесь ^: применяется к существительному (0 1 2)
5 4 3

   <: ^: i. n                NB. Здесь ^: применяется к глаголу (i.) справа
5 4 3 2 1

   */ <: ^: i. n
120

   f=: */ @ (<: ^: i.)
   f n
120

   f"0 i. n
1 1 2 6 24

   (f"0 = !) i. n
1 1 1 1 1

   <:/ ~ i. 5                NB. Таблица диады <:
1 1 1 1 1
0 1 1 1 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
```

Распаковать > 0 0 0 Больше

Распаковка обратна к упаковке, тоесть ><y дает y . В применении к открытому массиву (не имеющему упакованных элементов), распаковка не производит никакого эффекта. Распакованные атомы приводятся к общей размерности, как обсуждается в Разделе II B.

x>y есть 1 при x большем y, с учетом погрешности. См. Равно (=) для описания приближенных сравнений. Например:

```
   1 2 3 4 5  >  5 4 3 2 1
0 0 0 1 1
```

Погрешность t устанавливается >!.t .

Поскольку ранг глагола *распаковать* равен 0, он применяется к каждому атому своего аргумета по отдельности. Например:

```
   ]a=: 1 2 3;4 5 6;7 8 9
+-----+-----+-----+
|1 2 3|4 5 6|7 8 9|
+-----+-----+-----+

   >a
1 2 3
4 5 6
7 8 9
```

Результаты, имеющие разную размерность, подлежат выравниванию, как определено в Разделе II B. Например:

```
   (>1;2 3;4 5 6); (>'a';'bc';'def'); (<\i.4); (><\i.4)
+-----+---+--------------------+-------+
|1 0 0|a  |+-+---+-----+-------+|0 0 0 0|
|2 3 0|bc ||0|0 1|0 1 2|0 1 2 3||0 1 0 0|
|4 5 6|def|+-+---+-----+-------+|0 1 2 0|
|     |   |                    |0 1 2 3|
+-----+---+--------------------+-------+

   </~ i.5                   NB. Таблица диады <:
0 1 1 1 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
0 0 0 0 0

   1 < 1+10^-8+i.15          NB. Сравнение с погрешностью
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0

   1 <!.(0) 1+10^-8+i.15     NB. Точное сравнение (погрешность 0)
```

1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0

Потолок >. 0 0 0 Большее (Max)

>.y дает *потолок* для величины y , тоесть наименьшее целое, большее либо равное y . Таким образом:

```
   >. 4.6 4 _4 _4.6
5 4 _4 _4
```

Подразумеваемое при этом сравнение выполняется с погрешностью, как описано в Равно (=), которую можно установить союзом >.!.t . См. пол (<.) и McDonnell [10] для случая комплексных аргументов.

x>.y есть большее из x и y . Например:

```
   3>.4 _4
4 3

   >./7 8 5 9 2
9

   >./\7 8 5 9 2
7 8 8 9 9
```

Сравнение x = >. x определяет -- является ли x целым. Таким образом:

```
   Integer_test=: ] = >.      NB. См. определение вилки в Разделе II F.
   Integer_test 3 3.14 _5
1 0 1

   f=: = >.                   NB. Та же функция, но при помощи крючка.
   f 3 3.14 _5
1 0 1
```

Потолок >. y эквивалентен -<.-y . Другими словами, он является двойственным *полу* по отношению к (тоесть *под преобразованием*) перемене знака: >. ↔ <.&.- и <. ↔ >.&.- . Например:

```
   (<.&.- ; >.) 4.6 4 _4 _4.6
+---------+---------+
|5 4 _4 _4|5 4 _4 _4|
+---------+---------+
```

Плюс 1 `>: 0 0 0` Больше или Равно

`>:y` есть `y+1` . Например:

```
   >: 2 3 5 7
3 4 6 8
```

См. также Не `(-.)` .

`x>:y` дает 1 при `x` большем или равном `y`, с учетом погрешности.

См. Равно (=) для описания сравнений с погрешностью. `>:!.t` использует погрешность `t` .

```
   +: i. 6                    NB. Четные числа
0 2 4 6 8 10

   >: +: i. 6                 NB. Нечетные числа
1 3 5 7 9 11

   odds=: >:@+:@i.
   odds 10
1 3 5 7 9 11 13 15 17 19

   +/ odds 10
100

   (+/@odds , *:) 10          NB. Сумма n первых нечетных равна квадрату n
100 100

   >:/ ~ i. 5                 NB. Таблица диады >:
1 0 0 0 0
1 1 0 0 0
1 1 1 0 0
1 1 1 1 0
1 1 1 1 1
```

Знак "минус" и Бесконечность _

Символ _ перед цифрой обозначает отрицательное число (как в _3.4), а, будучи использованным самостоятельно, он обозначает бесконечность или отрицательную бесконечность (в случае __). Этот символ может быть использован и в именах, как обсуждается в Главе I и Разделе I Главы II.

Например:

```
   2 % 0                       NB. Два делить на ноль
_
   10 ^. 0                     NB. Логарифм нуля по основанию 10
__
   _2 _ 3 + 5
3 _ 8

   integer_test=: =<.          NB. Использование в имени
   integer_test 3 3.5
1 0
```

Несмотря на то, что -2 можно иногда использовать вместо _2, важно понимать, что первый случай есть применение глагола к числу 2, в то время как символ _ есть неотделимая часть представления числа, как точка есть неотделимая часть представления числа 8.9 .

Неопределенность `__.`

Неопределенность `_.` это числовой атом. Он призван помочь в работе с "не числами", полученными как данные из внешних источников, и должен быть удален из таких данных при первой возможности. Глагол `128!:5` проверяет наличие `_.`.

Способы создать `_.` следующие:

- непосредственный ввод `_.`
- `".`
- `3!:n`
- вызов DLL (базы данных, LAPACK, и т.д.)

Примитивы, которые только перемещают данные независимо от их значения, дают правильные и непротиворечивые результаты на содержащих `_.`, аргументах. К ним относятся `$ |. |: # , ,. [ ] { {. {: } }. }:`. Другие примитивы могут производить неверные или противеречивые результаты. В частности, диады `< <: = >: > ~: -: <. >.` дают противоречивые результаты на содержащих `_.` аргументах, так же как и `/: \:`, диады `i. i: e. -.`, монады `~. ~:`, а также определенные с их помощью глаголы.

Примитивы на аргументах, не содержащих `_.`, выдают `NaN error`, вместо того, чтобы произвести `_.` в качестве результата.

`x+y`	`x` and `y` бесконечности разных знаков
`x-y`	`x` and `y` бесконечности одного знака
`*y`	действительная и мнимая части `y` бесконечны
`x%y`	`x` и `y` бесконечны
`x^.y`	`x` и `y` 0 или бесконечность
`x!y`	`x` и `y` бесконечны
`m H. n`	similar to the dyad `p.`

`x p. y`	x и y вызывающие сложение бесконечностей противоположных знаков; например `1 _ _ p. _1`
`u T. n y`	подобно диаде `p.`

`0*_` , `0*__` , and `0%0` определены как `0` .

Бесконечность _: _ _ _ Бесконечность

_: есть *постоянная* функция, дающая бесконечный результат, тоесть _: y всегда _

_: есть *постоянная* функция, дающая бесконечный результат, тоесть x _: y всегда _

Например:

```
   y=: 1 2 3 4
   _: y
_
```

```
   _:"0 y                    NB. С рангом 0 применяется к каждому элементу
_ _ _ _
```

Другими постоянными функциями являются _9: , _8: и т.д. до 9: . В более общем случае, выражение x"r определяет постоянную функцию с рангом r, дающую постоянный результат x . Например:

```
   3.14"0 y
3.14 3.14 3.14 3.14

   3.14"1 y
3.14
```

Упомянутые постоянные функции могут быть записаны и как _"_ , _9"_ , 0"_ , 9"_ , и т.д.

Сопряжение + 0 0 0 Прибавить

+ y есть *комплексное сопряжение* величины y . Например, +3j4 есть 3j_4 .	+ определен как в элементарной арифметике и работает с комплексными числами обычным образом.

Комплексное число y, умноженное на свое сопряжение, дает действительное число, равное квадрату его модуля |y . Например:

```
   3j4 * 3j_4
25
```

Функция j. умножает свой аргумент на корень из минус единицы:

```
   ]i=: i. 5
0 1 2 3 4

   j. i
0 0j1 0j2 0j3 0j4

   ]y=: i + 2 * j. i
0 1j2 2j4 3j6 4j8

   +y
0 1j_2 2j_4 3j_6 4j_8

   y * +y
0 5 20 45 80

   %: y * +y
0 2.23607 4.47214 6.7082 8.94427

   |y
0 2.23607 4.47214 6.7082 8.94427
```

Комплексно сопряженное к y можно выразить и другим способом как (|y*y)%y . Например:

```
   (|y*y)%y
0 1j_2 2j_4 3j_6 4j_8
```

Действ. / Мнимая `+.  0 0 0` НОД (Или)

`+.y` выделяет действительную и мнимую части своего аргумента, возвращая их в виде двух-элементного списка. Например, `+.3j5` есть `3 5`, и `+.3` есть `3 0`.

`x+.y` есть *наибольший общий делитель* величин `x` и `y`. Если аргументы булевские (`0` или `1`), функция `+.` эквивалентна логическому *или*, а `*.` логическому *и*. Ограниченная подобным образом функция `-.`, есть логическое *не*.

```
   ]y=: i+2*j. i=: i.4
0 1j2 2j4 3j6

   +. y
0 0
1 2
2 4
3 6
```

При делении `x` и `y` на их наибольший общий делитель, результаты не имеют общего множителя, тоесть их НОД есть 1. Это соответствует сокращению дроби `x%y`. Например:

```
   x=: 24 [ y=: 60
   x;y;(x +. y);((x , y) % (x +. y))
+--+--+--+---+
|24|60|12|2 5|
+--+--+--+---+

   lff=: , % +.                Сокращает дробь
   x;y;(x lff y);(%/x lff y);(%/x,y);(+./x lff y)
+--+--+---+---+---+-+
|24|60|2 5|0.4|0.4|1|
+--+--+---+---+---+-+
```

Поскольку результат функций `=|` и `=<.` (тесты на положительные числа и целые числа, соответвтвенно) булевский, фраза `(=|)+.(=<.)` есть тест на положительное *или* целое число:

```
   (test=: (=|) +. (=<.)) _2 _2.4 3 3.5
1 0 1 1
```

Дуальность глаголов *или* и *и* может быть продемонстрирована следующим образом:

```
   d (+./ ; *.&.-./ ; *./ ; +.&.-./) d=: 0 1
+---+---+---+---+
|0 1|0 1|0 0|0 0|
|1 1|1 1|0 1|0 1|
+---+---+---+---+
```

Удвоить +: 0 0 0 Не-Или

`+:` у есть дважды у . Например:

```
   +: 3 0 _2
6 0 _4
```

х `+:` у есть отрицание х *или* у . Например, `0 +: 0` есть `1` .

Поскольку квадрат суммы двух чисел равен сумме их квадратов и их *удвоенного* произведения, следующие функции эквивалентны:

```
   f=: + * +
   g=: *:@[ + +:@* + *:@]
```

Например:

```
   x=: 7 6 3 [ y=: 6 5 3
   x (f ; g ; (f=g) ; (f-:g)) y
+----------+----------+-----+-+
|169 121 36|169 121 36|1 1 1|1|
+----------+----------+-----+-+
```

Поскольку область определения Не-Или ограничена числами ноль и единица, ее поведение можно полностью описать следующими таблицами истинности:

```
   d=: 0 1                    NB. Область определения Не-Или
   d +:/ d                    NB. Таблица истинности Не-Или
1 0
0 0

   d +./ d                    NB. Таблица Или
0 1
1 1

   -. d +./ d                 NB. Отрицание таблицы Или
1 0
0 0

   (+:&.-./~d) ; (*:/~d)      NB. Не-Или и Не-И дуальны относительно Не
+---+---+
|1 1|1 1|
|1 0|1 0|
+---+---+
```

Выделить Знак `*` `0 0 0` Умножить

`*y` дает `_1`, если величина `y` отрицательна; `0`, если она равна нулю; `1`, если положительна. В общем случае, `*y` есть точка пересечения единичного круга на комплексной плоскости с линией, проведенной из начала координат через точку `y` . Например:

```
   *_3 0 5 3j4
_1 0 1 0.6j0.8
```

Сравнение с нулем определено фразой `(y%|y)*t<:|y` и учитывает погрешность, где `t` обозначает ее (настраиваемую `*!.t`) величину.

`*` обозначает умножение, определенное в элементарной арифметике и расширенное для комплексных чисел обычным образом:

```
   t=:+.x,y [ x=:2j4 [ y=:5j3
   r=:-/*/t [ i=:+/ . * t
   (x,:y);t;r;i;(r j. i);(x*y)
+---+---+--+--+-----+-----+
|2j4|2 4|_2|26|_2j26|_2j26|
|5j3|5 3|  |  |     |     |
+---+---+--+--+-----+-----+
```

Выделение знака может быть полезно для произведения выборок. Например:

```
   * y=: _4 0 4
_1 0 1

   >:@* y
0 1 2

   f=: %:
   f ^: * " 0 y                NB. Обратный к f, Тождество, или f
16 0 2

   (* y) { ;:'Yes No Maybe'    NB. Выбор по индексу ({)
+-----+---+--+
|Maybe|Yes|No|
+-----+---+--+

   g=: <:`-:`+:@.*"0           NB. См. Сообразно (@.)
   g y
_8 _1 2
```

Диада `*` , примененная к списку и таблице, иллюстрирует важность *соответствия*, как обсуждается в Разделе II B:

```
   m=: i. 3 4 [ v=: 3 2 1
```

```
   m ; (v*m) ; (m*v) ; (+/ m*v) ; (v +/ . * m)
+---------+----------+----------+-----------+-----------+
|0 1  2  3|0  3  6  9|0  3  6  9|16 22 28 34|16 22 28 34|
|4 5  6  7|8 10 12 14|8 10 12 14|           |           |
|8 9 10 11|8  9 10 11|8  9 10 11|           |           |
+---------+----------+----------+-----------+-----------+
```

Модуль / Аргумент *. 0 0 0 НОК (И)

*.y производит двух-элементный список из длины и угла (в радианах) гипотенузы треугольника с основанием и высотой, равными действительной и мнимой частям y . Например, *. 3j4 есть 5 0.927295.

x*.y есть наименьшее общее кратное x и y . Для булевских аргументов (0 и 1) это соответствует логическому *и*. Таким образом:

```
   0 1 *./ 0 1
0 0
0 1
```

Проиллюстрируем некоторые свойства глагола Модуль / Аргумент. Покажем, в частности, что модуль (т.е. абсолютная величина) произведения двух комплексных чисел есть произведение их модулей, а аргумент произведения есть сумма их аргументов:

```
   (| ; *. ; r./@*.) y=: 3j4
+-+----------+---+
|5|5 0.927295|3j4|
+-+----------+---+

   x=: 2j_6
   *. x,y                                NB. Полярные координаты
6.32456 _1.24905
      5 0.927295

   f=: */@:({."1) , +/@:(}."1)           NB. Произведение по первому столбцу и сум
   f *. x , y
31.6228 _0.321751

   *. x * y                              NB. Модуль и аргумент произведения
31.6228 _0.321751
```

Наименьшее общее кратное есть произведение, деленное на НОД. Например:

```
   24 *. 60
120
   24 +. 60
12
   (24 * 60) % (24 +. 60)
120
```

В Квадрате *: 0 0 0 Не-И

*: у есть у в квадрате.	x *: у есть отрицание x *и* у . Например 0 *: 0 есть 1 .

Возведение в квадрат -- обратная операция к взятию квадратного корня. Например:

```
   *: ^: _1 (_2 _1 0 1 2)
0j1.41421 0j1 0 1 1.41421

   3 +&.*: 4                  NB. Гипотенуза треугольника со сторонами 3 и 4
5
```

Поскольку область определения функции Не-И ограничена нулем и единицей, ее поведение полностью описывается следующими таблицами истинности:

```
   d=: 0 1                    NB. Область определения Не-И
   d *:/ d                    NB. Таблица истинности Не-И
1 1
1 0

   d *./ d                    NB. Таблица И
0 0
0 1

   -. d *./ d
1 1
1 0
```

Не-И, Не И, и дуальный к Не-Или глагол совпадают, как показано ниже:

```
   (*:/~ ; -.@*./~ ; +:&.-./~) d
+---+---+---+
|1 1|1 1|1 1|
|1 0|1 0|1 0|
+---+---+---+
```

Поменять Знак

```
- 0 0 0
```

Вычесть

`-y` дает `y` с обратным знаком. Тоесть число, определенное как `0 - y`. Таким образом, `-2 0 _2` равно `_2 0 2`.

`-` определен как в элементарной арифметике, и расширен для комплексных чисел обычным образом.

Функция `-` обратна сама себе, тоесть `-^:_1` есть снова `-`.

Хоть `-2` можно часто использовать вместо `_2`, важно понимать, что первая запись есть применение функции к числу 2, в то время как `_` является неотделимой частью представления числа, как десятичная точка в `8.9`.

Не `-.` `0 _ _` Исключить

`-.y` равно `1-y` ; для булевских аргументов это соответствует отрицанию (Не); для вероятности, это дает вероятность отсутствия события.

`x-.y` включает все элементы `x` , кроме перечисленных в `y` .

Погрешность `t` можно настроить `-.!.t` .

Функция *исключить* применима к любой разумной паре агрументов. Например:

```
   (i. 9) -. 2 3 5 7
0 1 4 6 8

   'abcdefghij' -. 'aeiou'
bcdfghj

   ]m=: i. 4 5
 0  1  2  3  4
 5  6  7  8  9
10 11 12 13 14
15 16 17 18 19

   m -. 5 6 7 8 9
 0  1  2  3  4
10 11 12 13 14
15 16 17 18 19

   b=: <\ 'abcdefg'
   b
+-+--+---+----+-----+------+-------+
|a|ab|abc|abcd|abcde|abcdef|abcdefg|
+-+--+---+----+-----+------+-------+

   b -. 'abc';'abcde';'cba'
+-+--+----+------+-------+
|a|ab|abcd|abcdef|abcdefg|
+-+--+----+------+-------+

   2 3 4 5 -. 'abcdef'
2 3 4 5
```

Пополам `-:` `0 _ _` Совпадает

`-:y` есть половина `y` . Например:

```
   -: i. 5
0 0.5 1 1.5 2
```

`x -: y` дает 1 если аргументы совпадают по своим: размерности, упаковке и элементам (содержимому). Сравнение учитывает погрешность, см. Равно (=).

Глагол `-: !. t` производит сравнение с погрешностью `t` .

Например:

```
   x=: 0 1 2 3 4 5
   ,.&.> (] ; -: ; +:@-: ; (%&2) ; (2: %~ ])) x
+-+---+-+---+---+
|0|  0|0|  0|  0|
|1|0.5|1|0.5|0.5|
|2|  1|2|  1|  1|
|3|1.5|3|1.5|1.5|
|4|  2|4|  2|  2|
|5|2.5|5|2.5|2.5|
+-+---+-+---+---+

   x = +: -: x
1 1 1 1 1 1

   x -: +: -: x
1
```

Обратить % 0 0 0 Разделить

`%` y есть обратное к y , тоесть `1%y` . Например, `%4` ↔ `0.25` .	x `%` y есть частное от деления x на y , как определено в элементарной математике, за исключением того, что `0%0` есть `0` . См McDonnell [11], и конфигурацию средних столбца и строки в таблице ниже.

Мы продемонстрируем деление таблицами, используя функцию, генерирующую списки, симметричные вокруг нуля:

```
   ] a=: i: 3
_3 _2 _1 0 1 2 3

   (] ; *) |. a%/a
+-------------------------------------+--------------------+
|       _1 _1.5 _3  _  3  1.5         1|_1 _1 _1  1  1  1  1|
|_0.666667   _1 _2  _  2    1  0.666667|_1 _1 _1  1  1  1  1|
|_0.333333 _0.5 _1  _  1  0.5  0.333333|_1 _1 _1  1  1  1  1|
|        0    0  0  0  0    0         0| 0  0  0  0  0  0  0|
| 0.333333  0.5  1 __ _1 _0.5 _0.333333| 1  1  1 _1 _1 _1 _1|
| 0.666667    1  2 __ _2   _1 _0.666667| 1  1  1 _1 _1 _1 _1|
|        1  1.5  3 __ _3 _1.5         _1| 1  1  1 _1 _1 _1 _1|
+-------------------------------------+--------------------+

   6j2 ": |. a %/ a
 _1.00 _1.50 _3.00     _  3.00  1.50  1.00
 _0.67 _1.00 _2.00     _  2.00  1.00  0.67
 _0.33 _0.50 _1.00     _  1.00  0.50  0.33
  0.00  0.00  0.00  0.00  0.00  0.00  0.00
  0.33  0.50  1.00    __ _1.00 _0.50 _0.33
  0.67  1.00  2.00    __ _2.00 _1.00 _0.67
  1.00  1.50  3.00    __ _3.00 _1.50 _1.00
```

Использование форматирования в конце дает более читабельный результат, с шириной в шесть позиций на столбец и единообразными двумя цифрами после десятичной точки.

```
   |. a %/ x: a
  _1 _3r2 _3  _  3  3r2   1
_2r3   _1 _2  _  2    1  2r3
_1r3 _1r2 _1  _  1  1r2  1r3
   0    0  0  0  0    0    0
 1r3  1r2  1 __ _1 _1r2 _1r3
 2r3    1  2 __ _2   _1 _2r3
   1  3r2  3 __ _3 _3r2   _1
```

Обратить Матрицу

`%.  2 _ 2`

Разделить Матрицы

Если y невырожденная матрица, то `%.y` дает матрицу, обратную к ней. Например:

```
   mp=: +/ . *       NB. Произведение матриц
   (%. ; ] ; %. mp ]) i. 2 2
+--------+---+---+
|_1.5 0.5|0 1|1 0|
|   1   0|2 3|0 1|
+--------+---+---+
```

Более общо, глагол `%.y` определен через свой диадный случай с левым аргументом `=i.{:$y` (единичная матрица) или (эквивалентно) отношением `(%.y)mp x` ↔ `x %. y` .

Размерность `%.y` равна `|.$y` .

Векторный и скалярный случаи определены через матрицу `,.y`, но размерность результата равна `$y` .

Для ненулевого вектора y, `%.y` дает коллинеарный вектор с длиной, обратной длине y; это называется инверсией y по отношению к единичному кругу (или сфере). Таким образом:

```
   (%. ,: ] % %.) 2 3 4
0.0689655 0.103448 0.137931
       29       29       29
```

Если y невырожденная матрица, то `x%.y` равно `(%.y) mp x` . Более общо, если столбцы y линейно независимы, а `#x` и `#y` совпадают, то `x%.y` минимизирует разность:

```
   d=: x - y mp x %. y
```

в смысле того, что отклонение `+/d*+d` минимально. Скалярные и векторные y рассматриваются как матрица-столбец `,.y` .

Геометрически, `y mp x%.y` дает *проэкцию* вектора x на пространство столбцов y , ближайшую к x точку в векторном пространстве с базисом из столбцов y .

Обычно `%.` применяется для решения систем линейных уравнений и приближения функций многочленами, как в `c=: (f x)%. x ^ / i.4` .

Проиллюстрируем использование `%.` для приближения синуса многочленом, вычислив максимальный модуль ошибки по отношению к приближаемой функции:

```
   sin=: 1&o.                        NB. функция, которую мы будем приближать
   x=: 5 %~ i. 6
   c=: (sin x) %. x ^/ i.4           NB. Использование деления матриц
   ,.&.>@(] ; c"_ ; sin ; c&p. ; >./@:|@(sin-c&p.)) x
+---+-----------+--------+-----------+-----------+
|  0|_5.30503e_5|       0|_5.30503e_5|0.000167992|
```

```
|0.2|    1.00384|0.198669|   0.198826|           |
|0.4|  _0.018453|0.389418|   0.389321|           |
|0.6|  _0.143922|0.564642|   0.564523|           |
|0.8|           |0.717356|   0.717524|           |
|  1|           |0.841471|   0.841416|           |
+---+-----------+--------+-----------+-----------+
```

Квадратный Корень %: 0 0 0 Корень

`%: y` дает квадратный корень из `y` . Если величина `y` отрицательна, результатом является мнимое число. Например, `%:-4` ↔ `0j2` .

`x %: y` вычисляет корень `x`-й степени из `y` . Таким образом, `3%:8` равно `2` , а `2%:y` равно `%:y` . В общем случае, `x %: y` ↔ `y^%x` .

Например:

```
   y=: i. 7
   y
0 1 2 3 4 5 6

   2 %: y
0 1 1.41421 1.73205 2 2.23607 2.44949

   %: y
0 1 1.41421 1.73205 2 2.23607 2.44949

   r=: 1 2 3 4
   z=: r %:/ y
   z
0 1       2       3       4       5       6
0 1 1.41421 1.73205       2 2.23607 2.44949
0 1 1.25992 1.44225  1.5874 1.70998 1.81712
0 1 1.18921 1.31607 1.41421 1.49535 1.56508

   r ^~ z
0 1 2 3 4 5 6
0 1 2 3 4 5 6
0 1 2 3 4 5 6
0 1 2 3 4 5 6
```

См. *соответствие* в Разделе II B, и использовани

Экспонента ^ 0 0 0 В Степени

^y равно e^y, где e -- *число Эйлера* ^1 (≈ 2.71828). Обратным к ^ глаголом является *натуральный логарифм* ^. , тоесть y=^.^y и y=^^.y .

Монада x&^ обратна монаде x&^. . Например:

```
   10&^ 10&^. 1 2 3 4 5
1 2 3 4 5

   10&^. 10&^ 1 2 3 4 5
1 2 3 4 5
```

x^2 , x^3 и x^0.5 дают *квадрат*, *куб*, и *квадратный корень* x , соответственно.

В общем случае, x^y равно ^y*^.x, как для действительных, так и для комплексных чисел.

Для положительных целых y, фраза x ^ y эквивалентна */y # x; в частности, */ в применении к пустому списку дает 1, и x^0 равно 1 для любых x, включая 0 .

Союз *настроить*, в применении к ^ , производит лестницу (stope), определенную так, что x!.k n эквивалентно */x + k*i. n . В частности, ^!._1 представляет собой *спадающий факториал*.

Последний результат первого примера ниже иллюстрирует спадающий факториал, произведенный настройкой. См. Главу 5 в [14] для информации об использовании лестничных функций, лестничных многочленов (stope polynomial) и чисел Стирлинга в разностных методах:

```
   e=: ^ 1 [ x=: 4 [ y=: 0 1 2 3
   ,.&.> x (e"_ ; e&^@] ; ^ ; ^@(] * ^.@]) ; (]^]) ; ^!._1) y
+-------+-------+--+--+--+--+
|2.71828|      1| 1| _| 1| 1|
|       |2.71828| 4| 1| 1| 4|
|       |7.38906|16| 4| 4|12|
|       |20.0855|64|27|27|24|
+-------+-------+--+--+--+--+

   S2=: %.@S1=: (^!._1/~ %. ^/~) @ i. @ x:
   (S1;S2) 8
+--------------------------+-------------------+
|1 0  0  0  0   0    0     0|1 0 0 0 0  0  0   0|
|0 1 _1  2 _6  24 _120   720|0 1 1 1 1  1  1   1|
|0 0  1 _3 11 _50  274 _1764|0 0 1 3 7 15 31  63|
|0 0  0  1 _6  35 _225  1624|0 0 0 1 6 25 90 301|
|0 0  0  0  1 _10   85  _735|0 0 0 0 1 10 65 350|
```

```
|0 0  0  0  0   1  _15   175|0 0 0 0 0  1 15 140|
|0 0  0  0  0   0    1   _21|0 0 0 0 0  0  1  21|
|0 0  0  0  0   0    0     1|0 0 0 0 0  0  0   1|
+--------------------------+------------------+
```

S1 вычисляет числа Стирлинга первого рода (не опуская знак), а S2 числа Стирлинга второго рода. Их можно использовать для преобразования между лестничными и обычными многочленами. Заметьте, что x: включает неограниченную точность.

Натуральный Логарифм ^. 0 0 0 Логарифм

Натуральный логарифм (^.) обращает экспоненту ^ (т.е., y=^.^y и y=^^.y).

Логарифм по основанию x x^.y обращает степень (^) в том смысле, что y = x^.x^y и y = x^x^.y .

Некоторые свойства логарифмов иллюстрируются ниже:

```
   x=: 4 [ y=: 0 1 2 3
   (x^y);(x^.x^y);(x^.y);(x^x^.y)
+---------+-------+-----------------+-------+
|1 4 16 64|0 1 2 3|__ 0 0.5 0.792481|0 1 2 3|
+---------+-------+-----------------+-------+

   logtable=: ^./~@i.
   <6j2 ": logtable 6
+-------------------------------------+
|    _.  0.00  0.00  0.00  0.00  0.00|
|    __  0.00     _     _     _     _|
|    __  0.00  1.00  1.58  2.00  2.32|
|    __  0.00  0.63  1.00  1.26  1.46|
|    __  0.00  0.50  0.79  1.00  1.16|
|    __  0.00  0.43  0.68  0.86  1.00|
+-------------------------------------+
```

Первая производная логарифма есть обратная дробь. Например:

```
   ^. d. 1 y=: 0 1 2 3 4 5 6
_ 1 0.5 0.333333 0.25 0.2 0.166667

   % ^. d. 1 y
0 1 2 3 4 5 6
```

В Степени `u^:n _ _ _`

n может быть целым, упаковкой или герундием.

Целое. Глагол u применяется n раз. Бесконечная степень n повторяет вычисления до достижения неподвижной точки (предела) u . Например, `(2&o.^:_)1` дает `0.73908` , т.е. решение уравнения `y=Cos y` . Если n отрицательно, *обращение* `u^:_1` (см. ниже) применяется `|n` раз. Наконец, `u^:n y` для массива n получается, сборкой всех `u^:a y` (для всех атомов a из n) в результирующий массив.

Обращение используется и в u&.v , узнать его для данного v можно при помощи `v b. _1` . Повторного применения глагола можно достигнуть и при помощи C (&).

Упаковка. Если n упаковка, то ее содержимое должно быть атомом, тогда `u^:(<m)`

↔`u^:(i.m) y`	если m положительное целое
↔`u^:(i.k) y`	если m есть _ или '' , где k -- наименьшее положительное целое, такое что `(u^:(k-1) y) -: u^:k y`
↔`u^:_1^:(<|m) y`	если m отрицательно

Герундий. См. справа.

n может быть целым, упаковкой или герундием.

Целое или Упаковка. `x u^:n y` ↔ `x&u^:n y`

Герундий. (Сравните с применением герундия в наречии Заменяя }) `x u^:(v0`v1`v2)y` ↔ `(x v0 y)u^:(x v1 y) (x v2 y)`

`x u^:(  v1`v2)y` &harr; `x u^:([`v1`v2) y`

`  u^:(  v1`v2)y` &harr; `u^:(v1 y) (v2 y)`

Обращения (которые обычно, но не всегда, являются обратными друг другу глаголами в строгом смысле) можно разбить на шесть классов:

1. Функции, обращающие сами себя `+ - -. % %. |. |: /: [ ] C. p.`

2. Пары в следующих таблицах:

```
<  >                !     3 : '(-(!-y"_)%1e_3&* !"0 D:1
<: >:                      ])^:_^.y'
+. j./"1"_          3!:1  3!:2
+: -:               3!:3  3!:2
*. r./"1"_          \:    /:@|.
*: %:               ".    ":
^  ^.              j.    %&0j1
$. $.^:_1           o.    %&1p1
,: {.               p:    &pi;(n)
   ;@(,&'           q:    */
;: '&.>"1)          r.    %&0j1@^.
#. #:               s:    5&s:
                    u:    3&u:
                    x:    _1&x:

+~  -:
*~  %:
^~  3 : '(- -&b@(*^.) % >:@^.)^:_
    b=.^.y'"0
,~  <.@-:@# {. ]
,:~ {.
;~  >@{.
j.~ %&1j1
```

3. Очевидно обратимые связанные диады, такие как `-&3` и `10&^.` и `1 0 2&|:` и `3&|.` и `1&o.` и `a.&i.` , а так же `u@v` и `u&v` , если `u` и `v` обратимы.

4. Монады вида `v/\` и `v/\.` , где `v` один из `+ * - % = ~:`

5. Обращения, указанные явно союзом `:.`

6. Следующие случаи заслуживают особого внимания:
   `p:^:_1` `n` дает число простых чисел меньших `n`, обозначается в математике как π(n)
   `q:^:_1` есть `*/`

b&#^:_1 , где b булевский список, производит *расширение* массива (используемый при этом атом-заполнитель f можно настроить b&#^:_1!.f или #^:_1!.f)
a&#.^:_1 дает представление по основанию a
!^:_1 и !&n^:_1 и n&!^:_1 производят ожидаемые результаты
{= и i."1&1 являются обратными друг другу; они выполняют преобразование между представлениями перестановки в виде вектора целых чисел и в виде булевской матрицы

Пример 1:

```
   (] ; +/\ ; +/\^:2 ; +/\^:0 1 2 3 _1 _2 _3 _4) 1 2 3 4 5
+---------+-----------+------------+-------------+
|1 2 3 4 5|1 3 6 10 15|1 4 10 20 35|1  2  3  4  5|
|         |           |            |1  3  6 10 15|
|         |           |            |1  4 10 20 35|
|         |           |            |1  5 15 35 70|
|         |           |            |1  1  1  1  1|
|         |           |            |1  0  0  0  0|
|         |           |            |1 _1  0  0  0|
|         |           |            |1 _2  1  0  0|
+---------+-----------+------------+-------------+
```

Пример 2: Последовательность чисел Фибоначчи

```
   +/\@|.^:(i.10) 0 1
 0  1
 1  1
 1  2
 2  3
 3  5
 5  8
 8 13
13 21
21 34
34 55
   {. +/\@|.^:n 0 1x [ n=:128       NB. n-ное число фибоначчи
251728825683549488150424261
   {.{: +/ .*~^:k 0 1,:1 1x [ k=:7  NB. (2^k)-нное число фибоначчи
251728825683549488150424261
```

Пример 3: Итерация Ньютона

```
   -:@(+2&%)^:(0 1 2 3) 1
1 1.5 1.41667 1.41422
   -:@(+2&%)^:(_) 1
1.41421
   -:@(+2&%)^:a: 1
1 1.5 1.41667 1.41422 1.41421 1.41421
   %: 2
1.41421
```

Пример 4: Подгруппа, генерируемая Множеством Перестановок

```
   sg=: ~. @ (,/) @ ({"1/~) ^: _ @ (i.@{:@$ , ])
   sg ,: 1 2 3 0 4
0 1 2 3 4
1 2 3 0 4
2 3 0 1 4
3 0 1 2 4
   # sg 1 2 3 4 5 0 ,: 1 0 2 3 4 5
720
```

Пример 5: Транзитивное Замыкание

```
   x=: (#x)<. (#x),~x=: (i.20)+1+20 ?.@# 3
   (i.#x) ,: x
0 1 2 3 4 5 6 7  8  9 10 11 12 13 14 15 16 17 18 19 20
1 4 5 5 7 6 9 9 10 12 11 14 14 15 16 18 18 18 20 20 20
   {&x^:(<15) 0
0 1 4 7 9 12 14 16 18 20 20 20 20 20 20
   {&x^:a: 0
0 1 4 7 9 12 14 16 18 20
   x {~^:a: 0
0 1 4 7 9 12 14 16 18 20
```

Интерпретация: `x` представляет орграф с вершинами, пронумерованными как `i.#x` и ребрами от `i` к `i{x` . Например, выше присутствуют ребра: `0 1` , `1 4` , `2 5` , `3 5` и так далее. Тогда `{&x^:a:0` или `x{~^:a:0` вычисляет все узлы, достижимые из узла `0`.

Example 6: Транзитивное Замыкание

Каждая запись в файле начинается с байта, указывающего ее длину (исключая байт, ее кодирующий), после которого следует содержимое записи. Получив такой файл, глагол `rec` производит список упакованных записей.

```
rec=: 3 : 0
 n=. #y
 d=. _1 ,~ n<.1+(i.n)+a.i.y
 m=. d {~^:a: 0
 ((i.n) e. m) <;._1 y
)

randomfile=: 3 : 0
 c  =. 1+y ?@$ 255           NB. длины записей
 rec=. {&a.&.> c ?@$&.> 256  NB. содержимое записей
 (c{a.),&.> rec              NB. записи, вместе с их длинами
)

   boxed_rec=: randomfile 1000
   $ boxed_rec
1000

   file=: ; boxed_rec
   $ file
132045
```

```
   r=: rec file
   $r
1000

   r -: }.&.> boxed_rec
1
```

Последняя фраза проверяет отсутствие в результате `rec` байт, кодирующих длину записей.

В Степени u^:v _ _ _

Случай ^: с глаголом в качестве правого аргумента определен в терминах случая, когда правый аргумент существительное (u ^: n) следующим образом:

x u ^: v y ↔ x u^:(x v y) y
  u ^: v y ↔   u^:(v y) y

Например:

```
   x=: 1 3 3 1
   y=: 0 1 2 3 4 5 6
   x p. y
1 8 27 64 125 216 343

   x p. ^: (]>3:)"1 0 y
0 1 2 3 125 216 343

   a=: _3 _2 _1 0 1 2 3
   %: a
0j1.73205 0j1.41421 0j1 0 1 1.41421 1.73205

   * a
_1 _1 _1 0 1 1 1

   %: ^: * " 0 a
9 4 1 0 1 1.41421 1.73205

   *: a
9 4 1 0 1 4 9
```

Следующие монады эквивалентны. (См. пример ^ T. _ в определении Приближения Тейлора (T.) .)

```
   g=: u ^: p ^: _
   h=: 3 : 't=. y while. p t do. t=. u t end.'

   u=: -&3
   p=: 0&<
   (g"0 ; h"0) i. 10
+------------------------+------------------------+
|0 _2 _1 0 _2 _1 0 _2 _1 0|0 _2 _1 0 _2 _1 0 _2 _1 0|
+------------------------+------------------------+
```

Все разбиения целого (основывается на алгоритме, который предложил R.E. Boss 2005-07-05):

```
partition=: [: final ] (,new)@]^:[ (<i.1 0)"_
final    =: 0 <@-."1~ >@{:
new      =: [: <@; (-i.)@# cat&.> ]
cat      =: [ ,. (>: {."1) # ]

   partition 5
+-+---+---+-----+-----+-------+---------+
|5|4 1|3 2|3 1 1|2 2 1|2 1 1 1|1 1 1 1 1|
+-+---+---+-----+-----+-------+---------+
```

Взять Размерность `$ — 1 —` Придать Размерность

`$ y` дает размерность `y` , как определено в Разделе II A. Например, матрица 2x3 имеет размерность `2 3` , а размерность скаляра `3` есть пустой список (размерность которого `0`).

Ранг `y` есть `#@$ y` . Например:

```
   rank=: #@$
   (rank 3) , (rank ,3)
0 1

   (rank 3 4),(rank i. 2 3 4)
1 3
```

Размерность `x$y` равна `x,siy` , где `siy` -- размерность элемента `y`; `x$y` выдает ошибку в длине (length error), если `y` пустой и `x,siy` не содержит нуля. Например:

```
   y=: 3 4$'abcdefghijkl'
   y ; 2 2$ y
+----+----+
|abcd|abcd|
|efgh|efgh|
|ijkl|    |
|    |ijkl|
|    |abcd|
+----+----+
```

Этот пример иллюстрирует -- как составляется результат из *элементов* `y`; последняя 1-ячейка `(abcd)` свидетельствует о том, что продолжение является циклическим.

Настройка союзом (`$!.f`) позволяет сменить заполнитель на атом `f` , или, если `f` -- пустой вектор, на обычный заполнитель, определенный во Взять (`{.`).

Поскольку `x $ y` использует *элементы* из `y`, иногда может быть удобным разобрать правый аргумент, как в `x $ ,y`. Например (используя `y` , определенное выше):

```
   2 3 $ ,y
abc
def
```

Настройка может быть полезной для добавления нулей или пробелов. Например:

```
   8 $!.0 (2 3 4)
```

```
2 3 4 0 0 0 0 0

   ]z=: 8$!.'*' 'abc'
abc*****

   |. z
*****cba

   2 5$!.a: ;: 'zero one two three four five six'
+----+---+---+-----+----+
|zero|one|two|three|four|
+----+---+---+-----+----+
|five|six|   |     |    |
+----+---+---+-----+----+
```

`$.y` преобразует плотный массив в разреженный, и наоборот **`$.^:_1 y`** преобразует разреженный массив в плотный.

Тождества `f -: f&.$.` и `f -: f&.($.^:_1)` выполняются для любой функции `f` , за возможным исключением тех, которые (как в случае взятия избыточного количества элементов глаголом `{.`) используют элемент прореживания ("нулевой" элемент) в качестве заполнителя.

`0$.y` применяет `$.` или `$.^:_1` по возможности, тоесть преобразует плотный массив в разреженный, а разреженный в плотный.

`1$.sh;a;e` производит разреженный массив. `sh` указывает размерность. `a` указывает измерения, которые должны быть разрежены; можно использовать отрицательные индексы. `e` указывает "нулевой" элемент, его тип определяет тип данных в массиве. Аргумент можно сократить до `sh;a` (в качестве `e` тогда берется действительный `0`) или вообще до `sh` (тогда `a` считается `i.#sh` -- все измерения разрежены, а `e` действительным нулем).

`2$.y` дает разреженные измерения (`a`);
`(2;a)$.y` (пере-)определяет их;
`(2 1;a)$.y` дает число байт в `(2;a)$.y` ;
`(2 2;a)$.y` дает количество элементов в части `i` представления указанных разреженных измерений `a` (тоесть `#4$.(2;a)$.y`).

`3$.y` дает элемент прореживания (параметр `e`); **`(3;e)$.y`** переопределяет его.

`4$.y` дает матрицу индексов (часть `i`).

`5$.y` дает массив значений (часть `x`).

7$.y дает количество не-"нулевых" элементов в y , тоесть #4$.y или #5$.y .

8$.y удаляет все полностью "нулевые" значения и соответствующие строки в матрице индексов.

Обратный к n&$. глагол есть (-n)&$. .

Следующее подробное описание содержит разделы: Введение, Представление, Предположения, Дальнейшие Примеры, Линейная Алгебра Прореженных Массивов, и Текущее Состояние Реализации.

Введение

Ниже описывается расширение J для работы с массивами в которых "нули не занимают места". Для создания таких разреженных массивов определен новый глагол $. , а существующие примитивы расширены для работы с ними. Основные идеи можно иллюстрировать следующими примерами:

```
   ] d=: (?. 3 4$2) * ?. 3 4$100
0 55 79  0
0 39  0 57
0  0  0  0

   ] s=: $. d                              d            s
0 1 | 55
0 2 | 79                                    s
1 1 | 39                          "         "
1 3 | 57

   d -: s                         d   s
1

   o. s                           &pi;            s
0 1 | 172.788
0 2 | 248.186
1 1 | 122.522
1 3 | 179.071

   o. d                           &pi;            d
0 172.788 248.186       0
0 122.522       0 179.071
0       0       0       0

   (o. s) -: o. d
```

```
1

   0.5 + o. s
0 1 | 173.288
0 2 | 248.686
1 1 | 123.022
1 3 | 179.571

   <. 0.5 + o. s
0 1 | 173
0 2 | 248
1 1 | 123
1 3 | 179

   (<. 0.5 + o. s) -: <. 0.5 + o. d
1

   d + s                                  /
0 1 | 110
0 2 | 158
1 1 |  78
1 3 | 114

   (d + s) -: 2*s
1

   (d + s) -: 2*d
1

   +/ s
1 | 94
2 | 79
3 | 57

   +/"1 s
0 | 134
1 |  96

   |. s                                      ( . .                          )
1 1 | 39
1 3 | 57
2 1 | 55
2 2 | 79

   |."1 s
0 1 | 79
0 2 | 55
1 0 | 57
1 2 | 39

   |: s
1 0 | 55
1 1 | 39
2 0 | 79
3 1 | 57

   $ |: s
```

```
4 3

   $.^:_1 |: s                $.^:_1
 0  0 0
55 39 0
79  0 0
 0 57 0

   (|:s) -: |:d
1

   , s                                ;
1 | 55
2 | 79
5 | 39
7 | 57

   $ , s
12
```

Представление

Разреженный массив y может быть булевским, целым, действительным, комплексным, текстовым или упаковочным. Его внутренним представлением является sh;a;e;i;x , где:

- sh Размерность, $y . Элементы размерности должны быть меньше 2^31 , но их произведение может быть больше 2^31 .
- a Измерение(я), вектор отсортированных разреженных (индексированных) измерений.
- e Элемент прореживания ("ноль"). Элемент e также используется в качестве заполнителя, когда элементов массива недостаточно для выборки.
- i Индексы -- целая матрица индексов для разреженных измерений.
- x Значения -- (плотный) массив, содержащий (обычно) ненулевые ячейки из элементов неразреженных измерений, соответствующие индексам в матрице i .

Для разреженной матрицы s , использованной во введении,

```
   ] d=: (?. 3 4$2) * ?. 3 4$100
0 55 79  0
0 39  0 57
0  0  0  0

   ] s=: $. d
0 1 | 55
0 2 | 79
1 1 | 39
```

Размерность `3 4`; разреженные измерения `0 1`; элемент прореживания `0`; индексы -- первые две колонки чисел в отображении `s`, значения находятся в последней колонке.

Скаляры представляются плотно. Все примитивы принимают разреженные и/или плотные массивы в качестве аргументов (т.е. `sparse+dense` or `sparse$sparse`). Отображение разреженного массива состоит из отображения матрицы индексов (часть `i`), пустого столбца, столбца вертикальных линий, еще одного пустого столбца, и соответствующих ячеек значения (часть `x`).

Возможность установки произвольного элемента прореживания (не только нуля) позволяет замкнуть на разреженных массивах много больше функций (например `^y` или `10+y`), так что знакомые фразы будут производить знакомые результаты (например, `<.0.5+y` округляет до ближайшего целого).

Предположения

Для разреженного массива выполняются следующие предположения, проверяемые при каждом его отображении.

```
imax =: _1+2^IF64{31 63 наибольшее представимое целое
rank =: #@$             ранг
type =: 3!:0            тип данных

1 = rank sh             вектор
sh -: <. sh             целый
imax >: #sh             не больше imax элементов
(0<:sh) *. (sh<:imax)   ограничено 0 и imax

1 = rank a              вектор
a e. i.#sh              ограничено 0 и rank-1
a -: ~. a               элементы уникальны
a -: /:~ a              отсортированы

0 = rank e              атомны
(type e) = type x       имеет тот-же внутренний тип, что и x

2 = rank i              матрица
4 = type i              целое
```

`(#i) = #x`	столько строк, сколько элементов в `x`
`({:$i) = #a`	столько столбцов, сколько разреженных измерений
`(#i) <: */a{sh`	# строк ограничено произведением длин разреженных измерений
`imax >: */$i`	# элементов ограничено `imax`
`(0<:i) *. (i <"1 a{sh)`	`i` ограничено 0 и длинами разреженных измерений
`i -: ~.i`	строки уникальны
`i -: /:~ i`	строки отсортированы
`(rank x) = 1+(#sh)-#a`	ранг равен 1 плюс число плотных измерений
`imax >: */$x`	# элементов ограничено `imax`
`(}.$x)-:((i.#sh)-.a){s`	размерность элемента состоит из элементов размерностей, соответствующих плотным измерениям
`(type x) e. 1 2 4 8 16 32`	внутренний тип булевский, текстовый, целый, действительный, комплексный, или упакованный

Дополнительные Примеры

```
   ] d=: (0=?. 2 3 4$3) * ?. 2 3 4$100
46  0  0  0
 0 39  0  0
 0  0 46  0

 0  0  0  0
 0 60  0 62
 0  0 60 64
```

```
   ] s=: $. d
0 0 0 | 46
0 1 1 | 39
0 2 2 | 46
1 1 1 | 60
1 1 3 | 62
1 2 2 | 60
1 2 3 | 64
```

проредить `d` и присвоить `s`

```
   d -: s
1
```

совпадение не зависит от представления

```
   2 $. s
0 1 2
```

разреженные измерения

```
   3 $. s
```

элемент прореживания

```
0
```

```
   4 $. s                    матрица индексов; столбцы соотв. разреж. изм
0 0 0
0 1 1
0 2 2
1 1 1
1 1 3
1 2 2
1 2 3
```

```
   5 $. s                    соответствующие значения
46 39 46 60 62 60 64
```

```
   ] u=: (2;2)$.s            сделать разреженным только измерение 2
0 | 46  0  0
  |  0  0  0
  |
1 |  0 39  0
  |  0 60  0
  |
2 |  0  0 46
  |  0  0 60
  |
3 |  0  0  0
  |  0 62 64
```

```
   4 $. u                    матрица индексов
0
1
2
3
```

```
   5 $. u                    соответствующие значения
46  0  0
 0  0  0

 0 39  0
 0 60  0

 0  0 46
 0  0 60

 0  0  0
 0 62 64
```

```
   ] t=: (2;0 1)$.s          сделать разреженными измерения 0 1
0 0 | 46  0  0  0
0 1 |  0 39  0  0
0 2 |  0  0 46  0
1 1 |  0 60  0 62
1 2 |  0  0 60 64
```

```
   7 {. t                    взять
0 0 | 46  0  0  0
0 1 |  0 39  0  0
```

```
0 2 |  0  0 46  0
1 1 |  0 60  0 62
1 2 |  0  0 60 64

   $ 7 {. t
7 3 4

   7{."1 t                    взять с рангом
0 0 | 46  0  0  0 0 0 0
0 1 |  0 39  0  0 0 0 0
0 2 |  0  0 46  0 0 0 0
1 1 |  0 60  0 62 0 0 0
1 2 |  0  0 60 64 0 0 0

   0 = t
0 0 | 0 1 1 1
0 1 | 1 0 1 1
0 2 | 1 1 0 1
1 1 | 1 0 1 0
1 2 | 1 1 0 0

   3 $. 0 = t                 разреженный элемент 0=t есть 1
1

   +/ , 0 = t
17

   +/ , 0 = d                 результат не зависит от представления
17

   0 { t                      выбрать
0 | 46  0  0 0
1 |  0 39  0 0
2 |  0  0 46 0

   _2 (<1 2 3)}t              Заменяя
0 0 | 46  0  0  0
0 1 |  0 39  0  0
0 2 |  0  0 46  0
1 1 |  0 60  0 62
1 2 |  0  0 60 _2

   s=: 1 $. 20 50 1000 75 366
   $ s                        20 стран, 50 регионов, 1000 продавцов,
20 50 1000 75 366             75 продуктов, 366 дней в году

   */ $ s                     перемноженная размерность превышает 2^31
2.745e10

   r=: ?. 1e5 $ 1e6           доходы
   i=: ?. 1e5 5 $ $ s         где, за что и когда они получены
   s=: r (<"1 i)} s           занести в массив

   7 {. ": s                  первые 7 строк в отображении s
 0  0  20 48 150 | 395543     первая строка говорит: для страны 0, региона
```

```
 0  0  39 40  67 | 316198
 0  0  47 37 172 | 650782
 0  0  52 32 358 | 789844
 0  0  54 62  82 | 923413
 0  0  67 17 103 | 567367
 0  0  91 13 295 | 470919
```

продавца 20, продукта 48, дня 150,
доход был 395543

```
   +/ , s
|limit error
| +/    ,s
```

полный доход
выражение не сработало для ,s поскольку оно
потребовало бы вектор длины 2.745e10

```
   +/@, s
4.98338e10
```

полный доход
f/@, поддержана специальным кодом

```
   +/+/+/+/+/ s
4.98338e10

   +/^:5 s
4.98338e10

   +/^:_ s
4.98338e10

   +/ r
4.98338e10
```

полный доход

```
   +/"1^:4 s
 0 | 2.49298e9
 1 | 2.35118e9
 2 | 2.49324e9
 3 | 2.44974e9
 4 | 2.45138e9
 5 | 2.47689e9
 6 | 2.55936e9
 7 | 2.47153e9
 8 | 2.45907e9
 9 | 2.50249e9
10 | 2.52785e9
11 | 2.49482e9
12 | 2.57532e9
13 | 2.46509e9
14 | 2.54962e9
15 | 2.48942e9
16 | 2.50503e9
17 | 2.52147e9
18 | 2.50127e9
19 | 2.49603e9
```

полный доход по странам

```
   t=: +/^:2 +/"1^:2 s
```

полный доход по продавцам

```
   $t
1000

   7{.t
0 | 5.08254e7
```

```
1 | 5.61577e7
2 | 4.19914e7
3 | 5.90514e7
4 | 6.08208e7
5 | 4.10632e7
6 | 4.36616e7
```

Линейная Алгебра Разреженных Массивов

На данный момент, реализованы только умножение разреженных матриц и решение трёхдиагональных систем линейных уравнений. Например:

```
   f=: }. @ }: @ (,/) @ (,."_1 +/&_1 0 1) @ i.

   f 5                          индексы для трёхдиагональной матрицы 5 на
0 0
0 1
1 0
1 1
1 2
2 1
2 2
2 3
3 2
3 3
3 4
4 3
4 4

   s=: (?. 13$100) (<"1 f 5)} 1 $. 5 5;0 1
   $s
5 5
```

Фраза `1$.5 5;0 1` производит разреженный массив размерности `5 5` с разреженными измерениями `0 1`; `<"1 f 5` дает упакованные индексы; а `x (<"1 f 5)}y` заменяет `x` места в `y`, указанные индексами (замена вразброс).

```
   s
0 0 | 46
0 1 | 55
1 0 | 79
1 1 | 52
1 2 | 54
2 1 | 39
2 2 | 60
2 3 | 57
3 2 | 60
3 3 | 94
3 4 | 46
4 3 | 78
4 4 | 13
```

```
   ] d=: $.^:_1 s              плотное представление s
46 55  0  0  0
79 52 54  0  0
 0 39 60 57  0
 0  0 60 94 46
 0  0  0 78 13

   ] y=: ?. 5$80
66 75 79 52 54

   y %. s
0.352267 0.905377 0.00169115 0.764716 _0.434452

   y %. d                      ответы не зависят от представления
0.352267 0.905377 0.00169115 0.764716 _0.434452

   s=: (?. (_2+3*1e5)$1000) (<"1 f 1e5)} 1 $. 1e5 1e5;0 1

   $ s                         s -- матрица 1e5 на 1e5
100000 100000

   y=: ?. 1e5$1000

   ts=: 6!:2 , 7!:2@]          время и память на выполнение

   ts 'y %. s'
0.0550291 5.24358e6            0.056 секунд; 5.2 мегабайта (PIII 500 Mhz)
```

Текущее Состояние Реализации

По состоянию на 2005-12-17, разреженные массивы поддерживаются следующими примитивами:

```
=       =.      =:
< d     <.      <:
>       >.      >:
_       _.      _:

+       +.      +:
*       *.      *:
-       -.      -:
%       %. d    %:

^       ^.
$       $.      $:
~       ~.      ~:
|       |.      |:

        ..      .:
:       :.      ::
,       ,.      ,:
        ;.

#
```

```
!       !.      !:
/ m     /. d    /: m
\ m     \. m    \: m

[               [:
]
{ d     {.      {:
} d     }.      }:

"       ".      ": m
`               `:
@       @.      @:
&       &.      &:

e. d
i.
i:
j.
o.
r.
_9: to 9:

3!:0
3!:1
3!:2
3!:3
4!:55
```

Замечания:

- Разреженные массивы символов и упаковок пока не реализованы.
- Диада `%.` работает только для трёхдиагональных матриц.
- Упакованные левые аргументы для `|:` (диагональные срезы) пока не реализованы.
- Монады `f/` и `f/"r` реализованы только для `+ * >. <. +. *. = ~: ,` (и только булевских аргументов для `=` и `~:`); на измерении длины 2, монады `f/` и `f/"r` реализованы для любой функции.
- Монады `f/@,` (и `f/@:,` и `f/&,` и `f/&:,`) поддержаны специальным кодом.
- `{` и `}` принимают только следующие типы индексов: целые массивы, `<"1` на целых массивах, и скалярные упакованные индексы (соответственно, поэлементная индексация, индексация вразброс, и списки индексов `a0;a1;a2;...`); и (только `{`) разреженные массивы.

Вызвать Себя `$:  _ _ _`

`$:` обозначает самый длинный, содержащий его, глагол.

Пример 1:

```
   1:`(] * $:@<:)@.* 5
120
```

В этом выражении, Сообразно (`@.`) выбирает глагол `] * $:@<:` пока аргумент (уменьшаемый на единицу при каждом вызове) остается ненулевым. При нулевом аргументе, справа от `@.` получается ноль, и Сообразно выбирает постоянную функцию `1:` .

Если опустить `$:@` , вычисление будет выполнено только один раз:

```
   1:`(] * <:)@.* 5
20
```

Глагол Вызвать Себя приводит к тому, что вся функция выполняется снова после уменьшения аргумента на единицу.

Пример 2: В алгоритме быстрой сортировки (quicksort), из сортируемого списка случайно выбирается "центральный элемент". Результат тогда есть отсортированный список элементов, меньших центрального; присоединенный к списку элементов, равных центральному; присоединенный к отсортированному списку элементов, больших центрального.

```
   quicksort=: (($:@(#[)) ({~ ?@#)) ^: (1
```

Окружая `u~  _ ru lu` Зеркально

`u~ y` ↔ `y u y` . Например, `^~ 3` равно 27 , а `+/~ i. n` это таблица сложения.

`~` коммутирует или *меняет местами* аргументы: `x u~ y` ↔ `y u x` .

Некоторые применения Окружая и Зеркально иллюстрированы ниже:

```
   x=: 1 2 3 4 [ y=: 4 5 6
   x (,.@[ ; ^/ ; ^/~ ; ^/~@[ ; ]) y
+-+-------------+-------------+-----------+-----+
|1|  1    1    1|4 16  64  256|1  1  1   1|4 5 6|
|2| 16   32   64|5 25 125  625|2  4  8  16|     |
|3| 81  243  729|6 36 216 1296|3  9 27  81|     |
|4|256 1024 4096|             |4 16 64 256|     |
+-+-------------+-------------+-----------+-----+

   into=: %~
   (i. 6) % 5
0 0.2 0.4 0.6 0.8 1

   5 into i. 6
0 0.2 0.4 0.6 0.8 1

   from=: -~
   (i.6) - 5
_5 _4 _3 _2 _1 0

   5 from i.6
_5 _4 _3 _2 _1 0

   (x %/ y);(x %~/ y);(x %/~ y)
+-----------------+------------------+------------------+
|0.25 0.2 0.166667|      4       5   6|4   2 1.33333    1|
| 0.5 0.4 0.333333|      2     2.5   3|5 2.5 1.66667 1.25|
|0.75 0.6      0.5|1.33333 1.66667   2|6   3       2  1.5|
|   1 0.8 0.666667|      1    1.25 1.5|                  |
+-----------------+------------------+------------------+
```

Вызвать `m~` _

Если `m` -- имя, то `'m'~` эквивалентно `m` . Например:

```
   m=: 2 3 4
   'm'~
2 3 4

   m=: +/
   'm'~ 2 3 5 7
17

   m=: /
   + 'm'~ 2 3 5 7
17
```

Построить Множество ~. —

`~.y` выбирает *множество* элементов `y`, тоесть все его уникальные элементы. Например:

```
   y=: 3 3 $ 'ABCABCDEF'
   y;(~.y);(~.3);($~.3)
+---+---+-+-+
|ABC|ABC|3|1|
|ABC|DEF| | |
|DEF|   | | |
+---+---+-+-+
```

Точнее, множество вычисляется, выбирая первый элемент, вычеркивая из аргумента все приближенно (т.е. с погрешностью) равные ему, выбирая первый элемент из оставшихся, и т.д. Погрешность сравнения можно настроить (`!.`).

Если `f` требует значительных ресурсов для вычисления, определить `f y` можно, вычислив сначала `f~. y` (что дает все возможные значения функции), а затем, распределяя полученные значения по их окончательным позициям. Для осуществления последнего, достаточно умножить полученный вектор всех возможных значений на матрицу, производимую глаголом Найти В Себе (`=`). Например:

```
   f=: *:
   f y=: 2 7 1 8 2 8 1 8
4 49 1 64 4 64 1 64

   ,.&.>(~. ; f@~. ; = ; (f@~.(+/ .*)=) ; f)y
+-+--+--------------+--+--+
|2| 4|1 0 0 0 1 0 0 0| 4| 4|
|7|49|0 1 0 0 0 0 0 0|49|49|
|1| 1|0 0 1 0 0 0 1 0| 1| 1|
|8|64|0 0 0 1 0 1 0 1|64|64|
| |  |               | 4| 4|
| |  |               |64|64|
| |  |               | 1| 1|
| |  |               |64|64|
+-+--+--------------+--+--+
```

Наречие

```
   NUB=: 1 : 'x@~. +/ . * ='
   *: NUB y
4 49 1 64 4 64 1 64

   nubindex=: ~. i. ]
   (nubindex ; (nubindex { ~.)) y
```

```
+---------------+---------------+
|0 1 2 3 0 3 2 3|2 7 1 8 2 8 1 8|
+---------------+---------------+
```

Найти Множество `~:` `_ 0 0` Не Равно

`~:y` дает булевский список `b` , такой что `b#y` представляет собой множество элементов `y` . Например:

```
   ~: 'Mississippi'
1 1 1 0 0 0 0 0 1 0 0
```

`x~:y` дает `1` , если `x` не равно `y` с учетом погрешности. См. Равно (=).

Погрешность можно настроить, как в `~:!.t` .

Найденное множество можно выбрать следующим образом:

```
   y=: 8 1 8 2 8 1 7 2
   ~. y
8 1 2 7

   ~: y
1 1 0 1 0 0 1 0

   (~: y) # y
8 1 2 7

   y #~ ~: y
8 1 2 7
```

Диада `~:` применима к аргументам любого типа, но для булевских она называется *исключающее или*. Например:

```
   d=: 0 1
   d ~:/ d
0 1
1 0
```

Не Равно, Не *Поверх* Равно, и глагол, дуальный Равно по отношению к Не, совпадают:

```
   (~:/ ; -.@=/ ; =&.-./)~ d
+---+---+---+
|0 1|0 1|0 1|
|1 0|1 0|1 0|
+---+---+---+
```

Модуль | 0 0 0 Остаток

|y ↔ %:y*+y . Например:

```
   | 6 _6 3j4
6 6 5
```

Обычно, результат соответствует остатку от деления неотрицательного целого на положительное:

```
   3 | 0 1 2 3 4 5 6 7
0 1 2 0 1 2 0 1
```

Определение y-x*<. y % x+0=x расширяет область определения Остатка на случай нулевого левого аргумента, а также для случаев (конечных) отрицательных и дробных аргументов. Например:

```
   over =: ({. ,.@; }.)@":@,
   by   =: ' '&;@,.@[ ,. ]

   x=: 3 2 1 0 _1 _2 _3
   y=: 0 1 2 3 4 5 6 7 8

   x by y over x |/ y
+--+-----------------------+
|  |0  1  2  3  4  5 6  7  8|
+--+-----------------------+
| 3|0  1  2  0  1  2 0  1  2|
| 2|0  1  0  1  0  1 0  1  0|
| 1|0  0  0  0  0  0 0  0  0|
| 0|0  1  2  3  4  5 6  7  8|
|_1|0  0  0  0  0  0 0  0  0|
|_2|0 _1  0 _1  0 _1 0 _1  0|
|_3|0 _2 _1  0 _2 _1 0 _2 _1|
+--+-----------------------+
```

Чтобы для выражений типа (%3)|(2%3) результатом был точный ноль, Остаток использует погрешность, как показано в определении res ниже:

```
   res=: f`g@.agenda"0
      agenda=: ([ = 0:) +. (<. = >.)@S
         S=: ] % [ + [ = 0:
      f=: ] - [ * <.@S
      g=: ] * [ = 0:

   0.1 res 2.5 3.64 2 _1.6
0 0.04 0 0

   (,. ; res/~ ; |/~) a=: 2 -~ i.5
+--+-----------+-----------+
|_2| 0 _1 0 _1 0| 0 _1 0 _1 0|
|_1| 0  0 0  0 0| 0  0 0  0 0|
```

```
| 0|_2 _1 0  1 2|_2 _1 0  1 2|
| 1| 0  0 0  0 0| 0  0 0  0 0|
| 2| 0  1 0  1 0| 0  1 0  1 0|
+--+------------+------------+
```

Диада | применима и к комплексным числам. Кроме того, ее погрешность можно настроить (!.). Вычисление диады m&|@^ для целых аргументов избегает вычисления (потенциально больших) промежуточных результатов. Например: 2 (1e6&|@^) 10^100x

Перевернуть `|.` `_ 1 _` Сдвинуть

`|.` y переворачивает y поэлементно, изменяя порядок элементов на обратный. Например:

```
   |. t=: 'abcdefg'
gfedcba
```

Монада `|.!.f` выполняет *сдвиг вправо*, аналогично диаде с левым аргументом `_1`. Например:

```
   |.!.'#' t
#abcdef

   |.!.10 i.3 3
10 10 10
 0  1  2
 3  4  5
```

x`|.`y проворачивает (циклически) элементы в последовательных измерениях y на количество позиций, заданное соответствующими элементами x. Таким образом:

```
   1 2 |. i. 3 5
 7  8  9  5  6
12 13 14 10 11
 2  3  4  0  1
```

Фраза `x |.!.f y` выполняет *сдвиг*: элементы, которые при провороте вернулись бы обратно в массив, заменяются атомом `f`, если только `f` не пуст (`0=#f`), тогда они заменяются обычным заполнителем, определенным во Взять (`{.`):

```
   2 _2 |.!.'#'"0 1 t
cdefg##
##abcde
```

```
   y=: a.{~ (a. i. 'A') + i. 5 6

   () ; 2&|. ; _2&|. ; 2&|."1  ; 2&(|.!.'*'"1)) y
+------+------+------+------+------+
|ABCDEF|MNOPQR|STUVWX|CDEFAB|CDEF**|
|GHIJKL|STUVWX|YZ[\]^|IJKLGH|IJKL**|
|MNOPQR|YZ[\]^|ABCDEF|OPQRMN|OPQR**|
|STUVWX|ABCDEF|GHIJKL|UVWXST|UVWX**|
|YZ[\]^|GHIJKL|MNOPQR|[\]^YZ|[\]^**|
+------+------+------+------+------+

   () ; |. ; |."1 ; |.!.'*'"1  ; (2: |. ])) y
+------+------+------+------+------+
|ABCDEF|YZ[\]^|FEDCBA|*ABCDE|MNOPQR|
|GHIJKL|STUVWX|LKJIHG|*GHIJK|STUVWX|
|MNOPQR|MNOPQR|RQPONM|*MNOPQ|YZ[\]^|
|STUVWX|GHIJKL|XWVUTS|*STUVW|ABCDEF|
|YZ[\]^|ABCDEF|^]\[ZY|*YZ[\]|GHIJKL|
+------+------+------+------+------+

   1 _2 |. !. '*' 3{. y
```

```
**GHIJ
**MNOP
******
```

Транспонировать `|:` — Транспонировать `1 |:1`

`|:` переворачивает (переставляя в обратном порядке) измерения своего аргумента. Например:

```
   (] ; |:) i. 3 4
+---------+------+
|0 1  2  3|0 4  8|
|4 5  6  7|1 5  9|
|8 9 10 11|2 6 10|
|         |3 7 11|
+---------+------+
```

`x|:y` перемещает измерения `x` в конец. Если `x` упаковочный, в новое измерение объединяются элементы *диагонали* по совокупности измерений, перечисленных в каждом его элементе:

```
   y=: 3 4$'abcdefghijkl'
   y;(1 0|:y);(0|:y);((<0 1)|:y)
+----+---+---+---+
|abcd|aei|aei|afk|
|efgh|bfj|bfj|   |
|ijkl|cgk|cgk|   |
|    |dhl|dhl|   |
+----+---+---+---+
```

Например:

```
   y=: a.{~ (a. i. 'a') + i. 2 3 4
   z=: y;(2 1 |: y);((<2 1) |: y);(|: i. 4 5)
   z ,&< |:&.> z
+------------------------+------------------------+
|+----+---+---+---------+|+--+--+--+--------------+|
||abcd|aei|afk|0 5 10 15|||am|am|am| 0  1  2  3  4||
||efgh|bfj|mrw|1 6 11 16|||eq|bn|fr| 5  6  7  8  9||
||ijkl|cgk|   |2 7 12 17|||iu|co|kw|10 11 12 13 14||
||    |dhl|   |3 8 13 18|||  |dp|  |15 16 17 18 19||
||mnop|   |   |4 9 14 19|||bn|  |  |              ||
||qrst|mqu|   |         |||fr|eq|  |              ||
||uvwx|nrv|   |         |||jv|fr|  |              ||
||    |osw|   |         |||  |gs|  |              ||
||    |ptx|   |         |||co|ht|  |              ||
|+----+---+---+---------+||gs|  |  |              ||
|                        ||kw|iu|  |              ||
|                        ||  |jv|  |              ||
|                        ||dp|kw|  |              ||
|                        ||ht|lx|  |              ||
|                        ||lx|  |  |              ||
|                        |+--+--+--+--------------+|
+------------------------+------------------------+
```

По Минорам `u . v 2 _ _` Скалярно

Фразы `-/ . *` и `+/ . *` представляют собой *определитель* и *перманент* (определитель без знаков перестановок) квадратной матрицы. В более общем случае, фраза `u . v` определена в терминах рекурсивного разложения по минорам вдоль первого столбца, как показано ниже.

Для векторов и матриц, фраза `x +/ . * y` представляет собой *скалярное*, *внутреннее*, или *матричное* произведение, определенное в математике; аргументами могут быть и другие глаголы ранга 0, такие как `<.` и `*.`. В общем случае, `u . v` определено как `u@(v"(1+lv,_))`, эта фраза переведена на русский ниже.

Например:

```
   x=: 1 2 3 [ m=: >1 6 4;4 1 0;6 6 8
   det=: -/ . *
   mp=: +/ . *
   x ([ ; ] ; det@] ; mp ; mp~ ; mp~@]) m
+-----+-----+----+--------+-------+--------+
|1 2 3|1 6 4|_112|27 26 28|25 6 42|49 36 36|
|     |4 1 0|    |        |       | 8 25 16|
|     |6 6 8|    |        |       |78 90 88|
+-----+-----+----+--------+-------+--------+
```

Монада `u . v` определена следующим образом:

```
   DET=: 2 : 'v/@,`({."1 u . v $:@minors)@.(0<{:@$) @ ,. "2'
     minors=: }."1 @ (1&([\.))

   -/ DET * m
_112

   -/ DET * 1 16 64
49

   -/ DET * i.3 0
1

   +/ DET * m
320
```

Определение `u@(v"(1+lv,_))` для диадного случая можно пересказать словами следующим образом: `u` применяется к результату `v` на списках "ячеек левого аргумента" и правом аргументе *целиком*. Количество элементов в списках ячеек левого

аргумента должно соответствовать количеству элементов в правом аргументе. Таким образом, если `v` имеет ранги `2 3` , а размерности `x` и `y` равны `2 3 4 5 6` и `4 7 8 9 10 11`, то имеем `2 3` списков ячеек левого аргумента (размерности `4 5 6` каждая); и, если размерность результирующей ячейки есть `sr` , то размерность окончательного результата будет `2 3,sr` .

Четно , Нечетно u .. v u .: v

```
u .. v  &harr; (u + u&v) % 2:
u .: v  &harr; (u - u&v) % 2:
```

Чаще всего, в качестве v берется операция перемены знака, тогда f=: u .. v есть f=: (u + u&-) % 2: ; тоесть половина суммы u y и u -y . Получающаяся таким образом функция, является *четной* в том смысле, что f y ↔ f -y для любых y ; ее график симметричен относительно оси ординат. Подобно, u .: - является *нечетной* (f y ↔ -f-y), и ее график симметричен, относительно начала координат. Иногда в качестве v берется матричное транспонирование (|:) или другие монадные глаголы.

```
   y=: _2 _1 0 1 2
   1 2 3 4 5 & p. y           NB. Многочлен с четными и нечетными членами
57 3 1 15 129

   1 2 3 4 5 & p. .. - y      NB. Четная часть многочлена
93 9 1 9 93

   1 0 3 0 5 & p. y           NB. Многочлен только с четными степенями
93 9 1 9 93

   E=: .. -                   NB. Наречие Четно
   O=: .: -                   NB. Наречие Нечетно
   d=: 5j2&":@,.&.>           NB. Две цифры после точки в столбцах

   d (5&o. ; ^O ; 6&o. ; ^E ; ^ ; (^E + ^O) ; 2&o. ; ^@j.E) y
+-----+-----+-----+-----+-----+-----+-----+-----+
|_3.63|_3.63| 3.76| 3.76| 0.14| 0.14|_0.42|_0.42|
|_1.18|_1.18| 1.54| 1.54| 0.37| 0.37| 0.54| 0.54|
| 0.00| 0.00| 1.00| 1.00| 1.00| 1.00| 1.00| 1.00|
| 1.18| 1.18| 1.54| 1.54| 2.72| 2.72| 0.54| 0.54|
| 3.63| 3.63| 3.76| 3.76| 7.39| 7.39|_0.42|_0.42|
+-----+-----+-----+-----+-----+-----+-----+-----+

   m=: ?. 4 4 $ 9
   (] ; (] .. |:) ; (] .: |:)) m
+-------+-----------+-------------+
|3 8 8 4|  3 5 6 2.5|   0  3 2 1.5|
|2 0 2 7|  5 0 2   4|  _3  0 0   3|
|4 2 3 5|  6 2 3   6|  _2  0 0  _1|
|1 1 7 2|2.5 4 6   2|_1.5 _3 1   0|
+-------+-----------+-------------+
```

Явное Определение

```
m : n   — — —
```

Здесь m -- целое, обозначающее следующие варианты:

0 : n	существительное
1 : n	наречие
2 : n	союз
3 : n	глагол
4 : n	только диадный глагол
13 : n	если возможно, неявное представление глагола, иначе как 3 : n

n обычно 0, что обозначает ввод сценария (Раздел II H) с клавиатуры; в качестве n можно задать представление сценария в виде текстового списка s , упаковочного списка b=: , или текстовой таблицы t=:];._2 s . Таким образом:

```
f=: 3 : 0
a=: 2+b=. y ^ 2
a+a*b
:
x*x+y
)

   a=: b=: 19
   f 3
110

   a,b           NB. Изменилось только глобально присвоенное имя.
11 19
```

Как показано далее:

1. При использовании 3 : 0 , строка, состоящая из одного двоеточия, отделяет определения монадного и диадного случаев. Если такой строки нет, область определения диадного случая пуста.
2. Результатом явного определения является результат его последнего предложения, выполненного не в тестовом блоке; в случаях 3 : и 4 : этим результатом должно быть существительное. См. Управляющие Конструкции для определения блока.
3.

Присвоенные **=.** имена являются *локальными*; их новые значения не производят никакого эффекта как *вовне* данного определения, так и *внутри* других, вызванных им определений. Имена, присвоенные **=:** , являются глобальными (глобальное присваивание локальных имен не допускается). Присваивание локативов (с использованием =: или =.) всегда глобально.

4. Аргументы определения инициализируются =. . Для диадного случая это подобно:

```
  f=: 4 : 0
   x=. (левый   аргумент)
   y=. (правый аргумент)
   (остаток глагола)
  )
```

5. Имена x и y обозначают левый и правый аргументы. В определении союза можно ссылаться как на *его* левый и правый аргументы (используя u и v), так и на аргументы получающегося в результате его применения глагола (x и y); подобно, в определении наречия можно ссылаться как на его левый аргумент (используя u), так и на аргументы производимого глагола (x и y). Используя m вместо u (или n вместо v) можно ограничить класс соответствующего аргумента существительным. Например:

```
   conj=: 2 : '(u y)+ (v y)'
   mc=: 2 : 0
(u y)+(v y)
)

   dc=: 2 : 0                Диадный случай
:
(u y)+(v x)
)

   (!conj% 2 4 5);(!mc% 2 4 5);(1 2 3 !dc% 2 4 5)
+--------------+--------------+-------------+
|2.5 24.25 120.2|2.5 24.25 120.2|3 24.5 120.333|
+--------------+--------------+-------------+
```

Управляющие Конструкции. Последовательность исполнения предложений явного определения может быть изменена при помощи *ключевых слов*, таких как if. do. else. end. и while. . Например, глагол, находящий корень функции f по заданному двух-элементному списку чисел (локализующих корень), может быть определен и выполнен как:

```
   root=: 3 : 'm=.+/%#while.~:/y do.if.~:/*f b=.(m,{.)y do.y=.b else.y=.(m,{:)y e

   f=: 2 - *:
   b=: 1 10
   root b
```

```
1.41421
```

Для читабельности, это определение можно разбить на нескольких строк. Например так:

```
root=: 3 : 0
 m=. +/ % #
 while. ~:/y do.
  if. ~:/*f b=. (m,{.) y do.
   y=. b
  else.
   y=. (m,{:) y
  end.
 end.
 m y
)
```

Как видно из этого примера, слово `if.` и соответствующее слово `end.` отмечают начало и конец *управляющей конструкции*, так же как соответствующие `while.` и `end.` . Как иллюстрирует конструкция `if.` внутри конструкции `while.` , управляющие конструкции могут быть *вложены*. Слова `do.` и `else.` делят конструкцию `if.` на три простых блока, каждый из которых является предложением; а `do.` в конструкции `while.` делит ее на два блока: первый является простым предложением, а второй сам по себе является управляющей конструкцией `if.` . Таким образом, ключевые слова являются видом пунктуации.

Более детальное описание и примеры можно найти в разделе Управляющие Конструкции.

Монада / Диада

```
u : v   mu lv
    rv
```

Первый аргумент определяет монадный, а второй -- диадный случай глагола.

Например:

```
   y=: 10 64 100
   ^. y                         NB. Натуральный логарифм
2.30259 4.15888 4.60517

   10&^. y                      NB. Логарифм по основанию 10
1 1.80618 2

   log=: 10&^. : ^.
   log y
1 1.80618 2

   8 log y
1.10731 2 2.21462

   (^1) log y
2.30259 4.15888 4.60517

   LOG=: 10&$: : ^.             NB. Использование ссылки на себя
   LOG y
1 1.80618 2

   f=: %: : ($:@-)

   (f y),: 100 f y
3.16228 8 10
9.48683 6  0

   ABS=: | : [:

   ABS _4
4

   3 ABS _4
|valence error: ABS
|   3     ABS _4
```

Область определения диады ABS пуста, поскольку пуста область определения [: .

Обратный `u :. v  mu lu ru`

Результатом `u :. v` является глагол `u`, которому присвоено обращение `v` (используемое в качестве "обратного" глагола союзами `&.` и `^:`).

Например:

```
   y=: _4 0 4 3j4
   rp=: <@(%: , -@%:)"0        NB. Пары квадратных корней
   rp y
+--------+---+----+---------+
|0j2 0j_2|0 0|2 _2|2j1 _2j_1|
+--------+---+----+---------+

   I=: ^: _1
   rp I                        NB. Обращение не определено
rp^:_1

   rp I rp y
|domain error
|     rp I rp y

   inv=: *:@{.@,@>
   inv rp y
_4 0 4 3j4

   RP=: rp :. inv              NB. Глагол RP имеет обращение
   RP I RP y
_4 0 4 3j4

   rc=: <@(,: +)@(, -)@%:"0    NB. Корни и сопряженные
   rc y
+---------+---+----+----------+
| 0j2 0j_2|0 0|2 _2| 2j1 _2j_1|
|0j_2  0j2|0 0|2 _2|2j_1  _2j1|
+---------+---+----+----------+

   RC=: rc :. inv
   RC I RC y
_4 0 4 3j4
```

Ошибочный `u :: v _ _ _`

Результат `u :: v` есть результат `u`, если выполнение `u` завершается без ошибки; в противном случае, результатом является результат `v`.

Например:

```
   p=: 3 1 0 2                Вектор перестановки
   x=: 'ABCD'
   p{x
DBAC

   ]i=: A. p                  Индекс в упорядоченном списке всех переста
20

   i A. x                     Перестановка по индексу
DBAC

   q=: 3 1 1 0                Не перестановка
   q{x
DBBA

   A. q
|index error
|       A.q

   A=: A. :: (!@#)            При ошибке выдавать индекс больше максима
   A p
20

   A q
24

   24 A. x
|index error
|   24   A.x
```

Разобрать , Присоединить

`,y` дает список атомов `y` в "нормальном" порядке: результат упорядочен по элементам, элементам элементов, и т.д. Размерность результата есть `1$*/$ y` . Таким образом:

```
   y=: 2 4 $ 'abcdefgh'

   y
abcd
efgh

   ,y
abcdefgh
```

`x,y` добавляет элементы `y` в конец списка элементов `x` , перед этим:

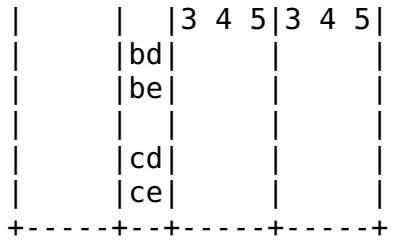
1. Если один из аргументов атом, ему придается размерность другого.
2. Аргументы приводятся к общему рангу (по меньшей мере 1), путем повторной *сборки* (`,:`) любого из них, имеющего меньший ранг, и
3. к общей размерности, путем добавления элементов заполнителя по схеме, описанной в Разделе II B.

Настройка (`,!.f`) позволяет задать заполнитель в виде атома `f` .

```
   ]a=: i. 2 3 3
 0  1  2
 3  4  5
 6  7  8

 9 10 11
12 13 14
15 16 17

   ,a
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

   ,"2 a
0  1  2  3  4  5  6  7  8
9 10 11 12 13 14 15 16 17
```

Следующие примеры иллюстрируют диадный случай:

```
   ('abc','de');('abc',"0/'de');(5 6 7,i.2 3);(7,i.2 3)
+-----+--+-----+-----+
|abcde|ad|5 6 7|7 7 7|
|     |ae|0 1 2|0 1 2|
```


3 4 5
3 4 5
bd
be
cd
ce

Разобрать Элементы ,. — — — Сшить

Если y атом, то ,.y дает 1 1$y , иначе результатом является ,"_1 y , тоесть таблица, полученная разборкой элементов y .

x,.y эквивалентно x,"_1 y . Другими словами, элементы x присоединяются к соответствующим элементам y .

Настройка (,.!.f) устанавливает в качестве заполнителя атом f .

Например:

```
   a=: i. 2 3 2
   ($,.3) ; (,.2 3 5 7 11) ; ($,.<'abcd') ; a ; (,.a)
+---+--+---+-----+-------------+
|1 1| 2|1 1| 0  1|0 1 2 3  4  5|
|   | 3|   | 2  3|6 7 8 9 10 11|
|   | 5|   | 4  5|             |
|   | 7|   |     |             |
|   |11|   | 6  7|             |
|   |  |   | 8  9|             |
|   |  |   |10 11|             |
+---+--+---+-----+-------------+
```

Следующие примеры иллюстрируют диадный случай:

```
   b=:3 4$'abcdefghijkl' [ c=:3 4$'ABCDEFGHIJKL'
   b ; c ; (b,.c) ; (b,c) ; a ; (a ,. |."1 a) ; (,/a) ; (,./a)
+----+----+--------+----+-----+-----+-----+---------+
|abcd|ABCD|abcdABCD|abcd| 0  1| 0  1| 0  1|0 1  6  7|
|efgh|EFGH|efghEFGH|efgh| 2  3| 2  3| 2  3|2 3  8  9|
|ijkl|IJKL|ijklIJKL|ijkl| 4  5| 4  5| 4  5|4 5 10 11|
|    |    |        |ABCD|     | 1  0| 6  7|         |
|    |    |        |EFGH| 6  7| 3  2| 8  9|         |
|    |    |        |IJKL| 8  9| 5  4|10 11|         |
|    |    |        |    |10 11|     |     |         |
|    |    |        |    |     | 6  7|     |         |
|    |    |        |    |     | 8  9|     |         |
|    |    |        |    |     |10 11|     |         |
|    |    |        |    |     | 7  6|     |         |
|    |    |        |    |     | 9  8|     |         |
|    |    |        |    |     |11 10|     |         |
+----+----+--------+----+-----+-----+-----+---------+
```

Собрать `,:` `_ _ _` Наклеить

`,:y` добавляет к `y` нулевое измерение из одного элемента, производя результат размерности `1,$y` . Тоесть:

```
   $ ,: 2 3 4
1 3
```

Вначале, атомному аргументу `x,:y` придается размерность другого аргумента (либо аргументы преобразуются в списки, если они оба атомные); затем, результаты собираются и присоединяются друг к другу, как в `(,:x),(,:y)` .

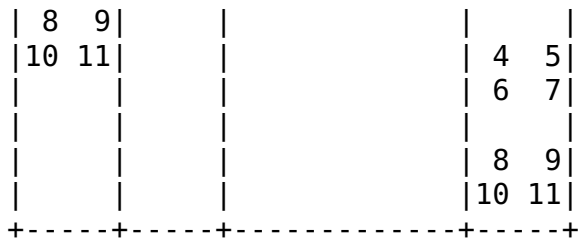
Настройка `(,.!.f)` устанавливает в качестве заполнителя атом `f` , либо, если `f` является пустым списком, обычный заполнитель, определенный во Взять (`{.`).

```
   s=: 3 [ v=: 2 3 4 [ m=: i. 3 3
   (,:s); ($,: s); (,:v); ($,:v); ($,:m); ($,:^:4 v)
+-+-+-----+---+-----+---------+
|3|1|2 3 4|1 3|1 3 3|1 1 1 1 3|
+-+-+-----+---+-----+---------+
```

Следующие примеры иллюстрируют диадные случаи глаголов Присоединить и Наклеить:

```
   a=: 'abcd' [ A=: 'ABCD' [ b=: 'abcdef'
   (a,A) ; (a,:A) ; (a,:b) ; (m,m) ; (m ,: m)
+--------+----+------+-----+-----+
|abcdABCD|abcd|abcd  |0 1 2|0 1 2|
|        |ABCD|abcdef|3 4 5|3 4 5|
|        |    |      |6 7 8|6 7 8|
|        |    |      |0 1 2|     |
|        |    |      |3 4 5|0 1 2|
|        |    |      |6 7 8|3 4 5|
|        |    |      |     |6 7 8|
+--------+----+------+-----+-----+
```

```
   t=: i. 3 2 2
   t ; (,/t) ; (,./t) ; (,:/t)
+-----+-----+------------+-----+
| 0  1| 0  1|0 1 4 5  8  9| 0  1|
| 2  3| 2  3|2 3 6 7 10 11| 2  3|
|     | 4  5|             |     |
| 4  5| 6  7|             | 0  0|
| 6  7| 8  9|             | 0  0|
|     |10 11|             |     |
```



Поломать ; _ _ _ Упаковать и Наклеить

;y составляет список распакованных элементов разобранного y . Настройка (;!.f) позволяет задать заполнитель в виде атома f .

x;y эквивалентно (<x),y если y упаковочный, и (<x),<y если нет.

```
   ]bv=: 1 2 3;4 5 6;7 8 9
+-----+-----+-----+
|1 2 3|4 5 6|7 8 9|
+-----+-----+-----+

   ;bv
1 2 3 4 5 6 7 8 9

   ]m=: >bv
1 2 3
4 5 6
7 8 9

   ;/ m
+-----+-----+-----+
|1 2 3|4 5 6|7 8 9|
+-----+-----+-----+

   (;/1 2 3 4 5) ,&< (;/i. 3 4)
+-----------+---------------------------+
|+-+-+-+-+-+|+-------+-------+---------+|
||1|2|3|4|5|||0 1 2 3|4 5 6 7|8 9 10 11||
|+-+-+-+-+-+|+-------+-------+---------+|
+-----------+---------------------------+

   ]txt=: '3 %: 4 ^. 5'
3 %: 4 ^. 5

   ]s=: ;: txt                NB. Разбивка на слова
+-+--+-+--+-+
|3|%:|4|^.|5|
+-+--+-+--+-+

   ;s
3%:4^.5

   (boxifopen=: <^:(< -: {:@;~)) 3 4
+---+
|3 4|
+---+
```

```
   (<3 4) = boxifopen <3 4
1
```

u;.0 y применяет u к y после переворота y вдоль каждого измерения; эквивалентно (0 _1 */$y) u;.0 y .

Разделитель 0{y (первый элемент y) отделяет интервалы элементов y ; фраза u;.1 y применяет u к каждому такому интервалу. Фраза u;._1 y отличается тем, что разделители не подвергаются обработке. Глаголы u;.2 и u;._2 используют в качестве разделителя *последний* элемент, отмечающий *концы* интервалов.

Монады u;.3 и u;._3 применяют u к покрытию "максимальными кубиками", точнее, они определены через свой диадный случай с левым аргументом ($$y)$<./$y .

m;.n y применяет последовательные глаголы из герундия m к разрезам y , продолжая m циклически сколько требуется.

x u;.0 y применяет u к прямоугольнику (или кубоиду) внутри y с вершиной в точке по индексу v=:0{x , и, с противоположной вершиной, определяемой так, что стороны кубоида равны |1{x , но, вдоль каждого измерения j для которого индекс j{v отрицателен, кубоид тянется *назад* из точки v . Наконец, порядок выбранных элементов меняется на обратный вдоль каждого измерения k , для которого k{1{x отрицательно. Если x -- вектор, вместо него берется матрица 0,:x .

В диадных случаях 1 , _1 , 2 , и _2 роль разделителей играют 1-цы в булевском векторе x ; пустой вектор x и ненулевое #y обозначает y целиком. Если x являтся атомом 0 или 1 , вместо него берется (#y)#x . В общем случае, булевский вектор >j{x определяет как будет проведен разрез вдоль измерения j ; если это атом, вместо него берется (j{$y)#>j{x .

u;.3 и u;._3 дают покрытия кубоидами (возможно и перекрывающимися). x u;._3 y применяет u к *полным* кубоидам размерами |1{x , начальные точки которых получаются многократным (поелементным) сложением направляющего вектора 0{x с самим собой. Если какой либо из размеров

бесконечный, вместо него берется длина соответствующего измерения со знаком. Аналогично `u;.0` , производится обращение вдоль всех измерений, для которых соответствующий размер в `1{x` отрицателен. Если `x` -- список, вместо него берется `1,:x` , производя полное покрытие кубоидами размера `x` . Случай `u;.3` отличается тем, что остаток с размером меньше `|1{x` включается в покрытие.

`x m;.n y` применяет последовательные глаголы из герундия `m` к разрезам `y` , расширяя `m` циклически сколько требуется.

Разрезы типов 0 и 3 имеют левый ранг 2, а типов 1 и 2 левый ранг 1.

```
   y=: 'worlds on worlds '
   (<;.2 y) ; ($;._2 y) ; (3 5$i.10) ; (+/ ;.1 (3 5$i.10))
+---------------------+-+---------+-----------+
|+-------+---+-------+|6|0 1 2 3 4|5 7 9 11 13|
||worlds |on |worlds ||2|5 6 7 8 9|0 1 2  3  4|
|+-------+---+-------+|6|0 1 2 3 4|           |
+---------------------+-+---------+-----------+

   1 0 1 0 0 <;.1 i.5 7
+--------------------+
|0 1 2  3  4  5  6   |
|7 8 9 10 11 12 13   |
+--------------------+
|14 15 16 17 18 19 20|
|21 22 23 24 25 26 27|
|28 29 30 31 32 33 34|
+--------------------+
   (';1 0 0 0 1 0 1) <;.1 i.5 7
+-----------+-----+--+
| 0  1  2  3| 4  5| 6|
| 7  8  9 10|11 12|13|
|14 15 16 17|18 19|20|
|21 22 23 24|25 26|27|
|28 29 30 31|32 33|34|
+-----------+-----+--+
```

```
   ('';1) <;.1 i.5 7
+--+--+--+--+--+--+--+
| 0| 1| 2| 3| 4| 5| 6|
| 7| 8| 9|10|11|12|13|
|14|15|16|17|18|19|20|
|21|22|23|24|25|26|27|
|28|29|30|31|32|33|34|
+--+--+--+--+--+--+--+
   (1 0 1 0 0;1 0 0 0 1 0 1) <;.1 i.5 7
+-----------+-----+--+
|0 1 2  3   | 4  5| 6|
|7 8 9 10   |11 12|13|
+-----------+-----+--+
|14 15 16 17|18 19|20|
|21 22 23 24|25 26|27|
|28 29 30 31|32 33|34|
+-----------+-----+--+

   x=:1 _2,:_2 3 [ z=: i. 5 5
   x ; (x ];.0 z) ; z
+-----+--------+--------------+
| 1 _2|11 12 13| 0  1  2  3  4|
|_2  3| 6  7  8| 5  6  7  8  9|
|     |        |10 11 12 13 14|
|     |        |15 16 17 18 19|
|     |        |20 21 22 23 24|
+-----+--------+--------------+

   (y=: a. {~ (a. i. 'a') + i. 4 4);(a=: 1 1 ,: 2 2)
+----+---+
|abcd|1 1|
|efgh|2 2|
|ijkl|   |
|mnop|   |
+----+---+

   (<;.3 y) ; ((($$y)$<./$y)<;.3 y) ; (a <;.3 y) ; <(a <;._3 y)
+---------------+---------------+------------+----------+
|+----+---+--+-+|+----+---+--+-+|+--+--+--+-+|+--+--+--+|
||abcd|bcd|cd|d|||abcd|bcd|cd|d|||ab|bc|cd|d|||ab|bc|cd||
||efgh|fgh|gh|h|||efgh|fgh|gh|h|||ef|fg|gh|h|||ef|fg|gh||
||ijkl|jkl|kl|l|||ijkl|jkl|kl|l||+--+--+--+-+|+--+--+--+|
||mnop|nop|op|p|||mnop|nop|op|p|||ef|fg|gh|h|||ef|fg|gh||
|+----+---+--+-+|+----+---+--+-+||ij|jk|kl|l|||ij|jk|kl||
||efgh|fgh|gh|h|||efgh|fgh|gh|h||+--+--+--+-+|+--+--+--+|
||ijkl|jkl|kl|l|||ijkl|jkl|kl|l|||ij|jk|kl|l|||ij|jk|kl||
||mnop|nop|op|p|||mnop|nop|op|p|||mn|no|op|p|||mn|no|op||
|+----+---+--+-+|+----+---+--+-+|+--+--+--+-+|+--+--+--+|
||ijkl|jkl|kl|l|||ijkl|jkl|kl|l|||mn|no|op|p||          |
||mnop|nop|op|p|||mnop|nop|op|p||+--+--+--+-+|          |
|+----+---+--+-+|+----+---+--+-+|           |          |
||mnop|nop|op|p|||mnop|nop|op|p||           |          |
|+----+---+--+-+|+----+---+--+-+|           |          |
+---------------+---------------+------------+----------+
```

Упаковать Слова ;: 1 — — Конечный Автомат

;:y есть список упакованных слов, выделенных из списка y в соответствии с правилами словообразования из Главы I и правилами относительно NB. . Функция неплохо работает и с обычным текстом.

Для подходящего левого аргумента x , результат x;:y эквивалентен ;:y . Таким образом:

```
mj=: 256$0                      NB. X другое
mj=: 1 (9,a.i.' ')}mj           NB. S пробел или табуляция
mj=: 2 ((a.i.'Aa')+/i.26)}mj    NB. A A-Z a-z кроме N B
mj=: 3 (a.i.'N')}mj             NB. N буква N
mj=: 4 (a.i.'B')}mj             NB. B буква B
mj=: 5 (a.i.'0123456789_')}mj   NB. 9 цифры и _
mj=: 6 (a.i.'.')}mj             NB. D .
mj=: 7 (a.i.':')}mj             NB. C :
mj=: 8 (a.i.'''')}mj            NB. Q кавычки

sj=: _2]\"1 }.".;._2 (0 : 0)
' X    S    A    N    B    9    D    C    Q ']0
 1 1  0 0  2 1  3 1  2 1  6 1  1 1  1 1  7 1 NB. 0 прб
 1 2  0 3  2 2  3 2  2 2  6 2  1 0  1 0  7 2 NB. 1 дрг
 1 2  0 3  2 0  2 0  2 0  2 0  1 0  1 0  7 2 NB. 2 а/ц
 1 2  0 3  2 0  2 0  4 0  2 0  1 0  1 0  7 2 NB. 3 N
 1 2  0 3  2 0  2 0  2 0  2 0  5 0  1 0  7 2 NB. 4 NB
 9 0  9 0  9 0  9 0  9 0  9 0  1 0  1 0  9 0 NB. 5 NB.
 1 4  0 5  6 0  6 0  6 0  6 0  6 0  1 0  7 4 NB. 6 чсл
 7 0  7 0  7 0  7 0  7 0  7 0  7 0  7 0  8 0 NB. 7 '
 1 2  0 3  2 2  3 2  2 2  6 2  1 2  1 2  7 0 NB. 8 ''
 9 0  9 0  9 0  9 0  9 0  9 0  9 0  9 0  9 0 NB. 9 ком
)

   x=: 0;sj;mj
   y=: 'sum=. (i.3 4)+/ .*0j4+pru 4'

   x ;: y
+---+--+-+--+---+-+-+-+-+-+---+-+---+-+
|sum|=.|(|i.|3 4|)|+|/|.|*|0j4|+|pru|4|
+---+--+-+--+---+-+-+-+-+-+---+-+---+-+
   (x ;: y) -: ;: y
1

   (5;sj;mj) ;: y
 0 _1 0 2 2 1
 1  0 2 2 2 0
 2  0 2 2 2 0
 3  0 2 0 1 2
```

x;:y реализует кон
(последовательную
спецификация авто
включающая табли
между состояниям
поток. Конечный ав
проблему распозна
входном потоке. Он
некоторого начальн
обрабатывает входн
последовательно по
элементу; Новое со
автомата определя
переходов, исходя
состояния и элемен
Потом автомат пер
обработке следующ
элемента. Детально

y может быть любы
x=.f;s;m;ijrd долж
упаковок, где ijrd
ijrd) можно опуст

f -- код функции, це
5 (объясняется ниж

m -- список, задающ
преобразование эле
потока: каждый упа
m содержит элемен
отображаемые на и
элемента; преобраз
образом входной по
my=. (y i.~;m) {
. Если y строка (те
может быть списко
неотрицательных ц
соответствующих ка
алфавита a. , тогда
будет отображен ка

```
 4  3 1 6 1 0
 5  3 1 1 0 3
 6 _1 0 0 1 1
 7  6 1 2 2 2
 8  7 2 6 1 0
 9  7 1 5 6 2
10  9 6 1 0 5
11 _1 0 5 6 1
12 11 6 0 1 4
13 12 1 0 1 2
14 13 1 0 1 2
15 14 1 1 0 3
16 _1 0 6 1 1
17 16 1 0 1 2
18 17 1 5 6 2
19 18 6 2 6 0
20 18 6 5 6 0
21 18 6 0 1 4
22 21 1 2 2 2
23 22 2 2 2 0
24 22 2 2 2 0
25 22 2 1 0 3
26 _1 0 5 6 1
```

Наконец, если m пу
-- числовой список)
входной поток my е

Массив s трехмерн
двух-колоночного м
неотрицательных ц
переходов и таблиц
размерность p,q,2
количество состоян
количество отображ
входного потока. То
q>#m если m есть сп
q>m если m список ш

ijrd представляет
целых (содержащий
элементов). i -- нач
счетчика итераций
индекс во входной
начальное состояни
начала текущего сл
определяющий дей
окончании входног
ниже). Требуется, ч
(0<:i)*.i<#y и (_1
. Если ijrd опущен
берется 0 _1 0 _1

x;:y проходит по в
отображенного вход
Пусть r -- текущее
индекс начала слов
(если ijr опущен) r
_1 . На итерации i
отображенный вход
c=.i{my и соответст
таблицы переходов
(список из двух цел
состояние при этом
вывода 1{e обознач

```
 0   нет вывода
 1   j=.i
 2   j=.i  [ ew(i
 3   j=._1 [ ew(i
 4   j=.i  [ ev(i
 5   j=._1 [ ev(i
```

```
                                                  6    остановиться

ew(i,j,r,c) ("emit
ошибку индексации
j равно _1 , иначе с
результирующий ма
о текущем слове, в
кодом функции f :

 0  <y{~j+i.i-j
 1  y{~j+i.i-j
 2  j,i-j
```

3

4

5

`ev(i,j,r,c)` ("emit vector") работает подобным образом, только текущее слово и все промежуточные присоединяются к предыдущему слову если предыдущий вывод был при помощи `ev` и состояние на тот момент было `r` .

После обработки последнего атома `my` , `i` принимает значение `#y` и осуществляется (если задано) действие, определяемое параметром `d` : Если `d=.3{ijrd` не отрицательно, выполняется дополнительная итерация с `c=.d` ; при отрицательном `d` и `j` не равном `_1` , выполняется
`ev(j,i,r,c)` .

Код функции `f=5` обозначает *трассировку*; в этом случае, результат `x;:y` -- матрица целых из 6-ти столбцов, где каждой итерации соответствует строка `i,j,r,c,s{~<r,c` . Эта матрица обычно имеет `#y` строк, но их может быть и меньше, если выполнение машины закончилось по коду 6 или были встречены коды выхода от 2 до 5, когда `j` было `_1` .

Таким образом `(0;s;m);:y` есть список упакованных элементов `y` , `(2;s;m);:y` есть матрица (из двух колонок) индексов и длин, и:

```
((0;s;m);:y) -: (2;s;m) (,"0@;: <;.0 ]) y
```

```
   s=: '*: @ -: @ i. 2 3'
   do=: ".
   do s
   0 0.25    1
2.25    4 6.25

   ;: s
+--+-+--+-+--+---+
|*:|@|-:|@|i.|2 3|
+--+-+--+-+--+---+

   ; ;: s
*:@-:@i.2 3

   p=: 'When eras die, their legacies/'
   q=: 'are left to strange police'
   r=: 'Professors in New England guard'
   s=: 'the glory that was Greece'

   ;: p
+----+----+---+-+-----+--------+-+
|When|eras|die|,|their|legacies|/|
+----+----+---+-+-----+--------+-+

   > ;: p,q
When
eras
die
,
their
legacies
/
are
left
to
strange
police

   |.&.;: p
/ legacies their , die eras When
```

Следующие примеры диады `;:` выделяют цитаты из текста. Кавычки отображаются в 1, а все остальные символы в 0; столбец 0 таблицы переходов соответствует "просто тексту" (не цитате), а столбец 1 цитате.

```
   sq=: 4 2 2$ 1 1 2 1  1 0 2 2  2 0 3 0  1 2 2 0
   <"1 sq
+---+---+
|1 1|2 1|
+---+---+
|1 0|2 2|
+---+---+
|2 0|3 0|
+---+---+
|1 2|2 0|
```

```
+---+---+
   ] y=: '''The Power of the Powerless'' by Havel and ''1984'' by Orwell'
'The Power of the Powerless' by Havel and '1984' by Orwell
   (0;sq;''''=a.) ;: y
+---------------------------+-------------+------+----------+
|'The Power of the Powerless'| by Havel and |'1984'| by Orwell|
+---------------------------+-------------+------+----------+
   (2;sq;''''=a.) ;: y
 0 28
28 14
42  6
48 10
   (3;sq;''''=a.) ;: y
6 3 6 2
   (4;sq;''''=a.) ;: y
 0 28 6
28 14 3
42  6 6
48 10 2

   sqx=: 4 2 2 $ 1 1 2 0  1 0 2 3  2 0 3 0  1 1 2 0
   <"1 sqx
+---+---+
|1 1|2 0|
+---+---+
|1 0|2 3|
+---+---+
|2 0|3 0|
+---+---+
|1 1|2 0|
+---+---+
   (1;sqx;''''=a.) ;: y
 by Havel and  by Orwell

   f=: (1;sqx;''''=a.)&;:
   g=: (+: ~:/\)@(''''&=) # ]
   (f -: g) y
1
```

Подобный автомат можно использовать и для извлечения только строк в кавычках (цитат). При этом, для предотвращения ситуации, когда последняя незакрытая кавычка считается началом цитаты, необходимо (при помощи параметра `ijrd`) изменить интерпретацию конца строки (не считая его кавычкой, оканчивающей неоконченную цитату):

```
   ] y=: '''Preposterous!''  He couldn''t go on.'
'Preposterous!'  He couldn't go on.
   sq=: 4 2 2$ 1 1 2 1  1 0 2 1  2 0 3 0  1 3 2 0
   (0;sq;''''=a.) ;: y
+--------------+---------+
|'Preposterous!'|'t go on.|
+--------------+---------+
   (0;sq;(''''=a.);0 _1 0 0) ;: y
+--------------+
|'Preposterous!'|
```

```
+---------------+
```

Лабораторные "Sequential Machines" и "Huffman Coding" содержат другие примеры использования конечных автоматов.

Сосчитать `#  _ 1 _` Копировать

`#y` дает количество элементов в `y` . Тоесть:

```
   (#'');(#'a');(#'octothorpe')
+-+-+--+
|0|1|10|
+-+-+--+

   (#3);(#,3);(# 3 4)
+-+-+-+
|1|1|2|
+-+-+-+

   (#i.4 5 6);(#$i.4 5 6)
+-+-+
|4|3|
+-+-+
```

Если аргументы имеют одинаковое число элементов, то `x#y` копирует `+/x` элементов из `y`, при этом элемент `i{y` повторется `i{x` раз. Иначе, если один из аргументов атом, перед копированием он повторяется столько раз, сколько элементов в другом аргументе.

Комплексный левый аргумент `a j. b` копирует `a` элементов, после которых идет `b` заполнителей. Заполнитель можно настроить при помощи `#!.f` .

Копирование иллюстрируется следующими примерами:

```
   0 1 2 3 4 5 # 0 1 2 3 4 5
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5

   t=: 3 4 $'abcdefghijkl' [ n=: i. 3 4
   t ; n ; (3 0 1 # t) ; (3 0 1 # n) ; (3 1 4 2 #"1 t)
+----+---------+----+---------+----------+
|abcd|0 1  2  3|abcd|0 1  2  3|aaabccccdd|
|efgh|4 5  6  7|abcd|0 1  2  3|eeefgggghh|
|ijkl|8 9 10 11|abcd|0 1  2  3|iiijkkkkll|
|    |         |ijkl|8 9 10 11|          |
+----+---------+----+---------+----------+

   k=: 2j1 0 1j2
   (k # t);(k # n);(k  #!.'*' t);(k #!.4 n)
+----+---------+----+---------+
|abcd|0 1  2  3|abcd|0 1  2  3|
|abcd|0 1  2  3|abcd|0 1  2  3|
|    |0 0  0  0|****|4 4  4  4|
|ijkl|8 9 10 11|ijkl|8 9 10 11|
|    |0 0  0  0|****|4 4  4  4|
|    |0 0  0  0|****|4 4  4  4|
+----+---------+----+---------+
```

Из Двоичной `#. 1 1 1` Из N-ричной

`#.y` дает значение y по основанию два, тоесть `2#.y` . Например:

```
   #. 1 0 1 0
10

   #. 2 3$ 0 0 1,1 0 1
1 5
```

`x#.y` дает взвешенную сумму элементов `y` ; тоесть `+/w*y` , где `w` -- список частичных произведений `*/\.}.x,1` . Атомному аргументу придается размерность другого аргумента.

```
   ]a=: i. 3 4
0 1  2  3
4 5  6  7
8 9 10 11

   10 #.a
123 4567 9011

   8 #. a
83 2423 4763

   ]time=: 0 1 3,1 1 3,:2 4 6
0 1 3
1 1 3
2 4 6

   x=: 24 60 60
   x #. time
63 3663 7446

   x,1
24 60 60 1

   ]w=: */\.}. x,1
3600 60 1

   w *"1 time
   0  60 3
3600  60 3
7200 240 6

   +/"1 w *"1 time
63 3663 7446

   w +/@:* "1 time
63 3663 7446

   c=: 3 1 4 2 [ y=: 0 1 2 3 4 5
   c p. y
```

Многочлен с коэффициентами `c`

```
3 10 37 96 199 358

   y #."0 1 |.c
3 10 37 96 199 358
```

В Двоичную `#: _ 1 0` В N-ричную

`#: y` дает двоичное представление `y` , как и `(m#2)#:y` , где `m` -- максимальное количество цифр, необходимых для представления атомов `y` по основанию `2` . Например:

```
   i. 8
0 1 2 3 4 5 6 7

   #: i. 8
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
```

В простых случаях `r&#:` является обратным к `r&#.` . Тоесть:

```
   r=: 24 60 60
   r #: r #. 2 3 4
2 3 4
```

Но если `r #. y` превышает `(*/r)-1` (наибольшее целое, представимое по основанию `r`), то в качестве результата `r#:y` берется остаток от деления на `*/r` . Например:

```
   r #: r #. 29 3 4
5 3 4
```

Представление в произвольной системе счисления (аналогично представлению в двоичной, осуществляемому монадным случаем глагола `#:`) можно получить следующим образом:

```
   ndr=: 1 + <.@^.              количество требуемых цифр

   10 ndr y=: 9 10 11 100 99 100
1 2 2 3 2 3

   (y#:~10 #~ >./10 ndr y);(y#:~8 #~ >./8 ndr y)
+-----+-----+
|0 0 9|0 1 1|
|0 1 0|0 1 2|
|0 1 1|0 1 3|
|1 0 0|1 4 4|
|0 9 9|1 4 3|
|1 0 0|1 4 4|
+-----+-----+

   (10&#.^:_1 ; 8&#.^:_1) y
+-----+-----+
|0 0 9|0 1 1|
|0 1 0|0 1 2|
|0 1 1|0 1 3|
|1 0 0|1 4 4|
|0 9 9|1 4 3|
|1 0 0|1 4 4|
+-----+-----+
```

Факториал ! 0 0 0 Количество Сочетаний

Для неотрицательного целого y, определением является */>:i.y . В общем случае, !y есть Γ(1+y) (Гамма функция). Тоесть:

```
   (*/1 2 3 4 5) , (!5)
120 120

   ]x=: 2 %~ 3 -~ i. 2 4
_1.5 _1 _0.5 0
 0.5  1  1.5 2

   !x
_3.54491 _ 1.77245 1
0.886227 1 1.32934 2

   ]fi=:!^:_1(24 25 2.1 9876)
4 4.02705 2.05229 7.33019

   ! fi
24 25 2.1 9876
```

Для неотрицательных целых x!y есть количество способов, которыми можно выбрать x обьектов из y . В общем случае, x!y есть (!y)%(!x)*(!y-x) . Тоесть:

```
   3!5
10
   (!5)%(!3)*(!5-3)
10
   1j2 ! 3.5
8.64269j16.9189

   ]y=:2&!^:_1 (45 4.1 30 123)
10 3.40689 8.26209 16.1924
   2 ! y
45 4.1 30 123

   ]x=:!&10^:_1 (2.5 45)
0.3433618 2
   x ! 10
2.5 45
```

Первая таблица ниже иллюстрирует связь между диадой ! и таблицей биномиальных коэффициентов; последние две иллюстрируют связь с *фигурными числами* (*figurate numbers*):

```
   h=: 0,i=: i.5 [ j=: -1+i.5 [ k=: 5#1
   tables=: (,.h);(i,i!/i);(j,i!/j);(k,i(+/\^:)k)
   format=: ({. ,:&< }.)@":&.>
   format tables
+---+----------+------------------+-------------+
|+-+|+---------+|+----------------+|+------------+|
||0|||0 1 2 3 4|||_1 _2  _3  _4  _5|||1 1  1  1  1||
|+-+|+---------+|+----------------+|+------------+|
||0|||1 1 1 1 1||| 1  1   1   1   1|||1 1  1  1  1||
||1|||0 1 2 3 4|||_1 _2  _3  _4  _5|||1 2  3  4  5||
||2|||0 0 1 3 6||| 1  3   6  10  15|||1 3  6 10 15||
||3|||0 0 0 1 4|||_1 _4 _10 _20 _35|||1 4 10 20 35||
||4|||0 0 0 0 1||| 1  5  15  35  70|||1 5 15 35 70||
|+-+|+---------+|+----------------+|+------------+|
+---+----------+------------------+-------------+
```

Все фигурные числа порядка ноль равны единице; числа более высоких порядков, получаются последовательным применением

префиксных частичных сумм (тоесть сумм по префиксам +/\). Фигурные числа второго порядка называются *треугольными числами*, и получаются из префиксных частичных сумм целых чисел, начиная с единицы.

`m comb n` генерирует все выборки `m` чисел из `i.n`:

```
   seed=: [: i.@(,&0)&.> <:@- {. 1:
   cf  =: i.@# ,.&.> ,&.>/\.@:(>:&.>)
   comb=: [: ; [ cf@[&0 seed

   3 comb 5
0 1 2
0 1 3
0 1 4
0 2 3
0 2 4
0 3 4
1 2 3
1 2 4
1 3 4
2 3 4
```

Настроить `!.`

Этот союз изменяет некоторые глаголы способами, указанными в их описаниях. Например, `=!.t` -- соотношение равенства с погрешностью `t` , а `^!.r` -- *факториальная функция* определенная так, что `x ^!.r n` равно `*/x + r * i. n` . Следовательно, `^!._1` -- *спадающий факториал*.

Настроить можно следующие глаголы (произведя их *варианты*). Далее монадный случай описан до разделителя, диадный после:

`< <: > >: +. *. -. -: | E. i. i:`	• Погрешность
`<. >. * ~.`	Погрешность •
`#:`	• Погрешность
`= ~: #: e.`	Погрешность • Погрешность
`^ p.`	• Лестничная функция и соотв. многочлен
`$ |. , ,. ,: # {.`	• Заполнитель
`":`	Точность при выводе •

Внешний !:

Этот союз используется для взаимодействия с операционной системой, клавиатурой (как устройством ввода) и экраном (как устройством вывода). Она так же используется для доступа к различным возможностям, не относящимся непосредственно к языку, таким как: установка способа отображения функций, определение класса имен (существительное, глагол, наречие, или союз), получение списка всех имен заданного класса, и т.д.

Приложение A перечисляет все возможности применения Внешнего союза.

```
   (mean=: +/ % #) a=: 2 3 5 7 11 13
6.83333

   mean
+/ % #

   9!:3 (4)
   mean
  +- / --- +                  Древовидное отображение глагола
--+- %
  +- #

   9!:3 (2 4 5)
   mean
+-----+-+-+                   Упаковочное отображение
|+-+-+|%|#|
||+|/|| | |
|+-+-+| | |
+-----+-+-+
  +- / --- +                  Древовидное отображение
--+- %
  +- #
+/ % #                        Линейное отображение

   4!:0 'a';'mean'            Классы имен (существ. -- 0, глагол -- 3)
0 3

   4!:1 (3)                   Список имен класса 3
+----+
|mean|
+----+
```

Между m/ u/ _ _ _ Таблично

u/y вставляет диаду u между элементами y . Тоесть:

```
   m=: i. 3 2
   m;(+/m);(+/"1 m);(+/2 3 4)
+---+---+-----+-+
|0 1|6 9|1 5 9|9|
|2 3|   |     | |
|4 5|   |     | |
+---+---+-----+-+
```

m/y вставляет последовательные глаголы из герундия m между элементами y, расширяя m циклически сколько требуется. Тоесть, +`*/i.6 есть 0+1*2+3*4+5 .

Если x и y числовые списки, тогда x */ y дает их таблицу умножения. Тоесть:

```
   1 2 3 */ 4 5 6 7
 4  5  6  7
 8 10 12 14
12 15 18 21
```

В общем случае, каждая ячейка x применяется ко всему y . Тоесть x u/ y эквивалентно x u"(lu,_) y где lu есть левый ранг u .

В тензорном анализе */ называется *прямым (внешним) произведением*.

Если y не имеет элементов (т.е., 0=#y), u/y производит *нейтральный* или *тождественный элемент* функции u . Нейтральный элемент функции u есть значение e такое, что x u e ↔ x или e u x ↔ x, для любого x в области определения (или некоторой ее важной под-области, такой как булевская) u . Это определение (вставки между элементами аргумента, *не содержащего* элементов) расширяет тождества разбиений вида u/y ↔ (u/k{.y) u (u/k}.y) на случаи k e. 0,#y .

Тождественная функция u есть функция ifu , такая что ifu y ↔ u/y if 0=#y . Используются следующие тождественные функции:

Тождественная функция	**Для**
0 $~ }.@$	< > + - +. ~: | (2 4 5 6 b.)
1 $~ }.@$	= <: >: * % *. %: ^ ! (1 9 11 13 b.)
_ $~ }.@$	<.
__ $~ }.@$	>.
(v^:_1 ifu$0) $~ }.@$	u&.v
i.@(0&,)@(2&}.)@$	,

```
/:@{.                    C.  {
=@/:@{.                  %.  +/ . *
ifu@#                    u/
```

Диагонально `m/.` `u/.` `_ _ _` По Ключу

`u/.y` применяет `u` к каждой анти-диагонали матрицы `y` . Например:

```
   i.3 4
0 1  2  3
4 5  6  7
8 9 10 11

   </. i.3 4
+-+---+-----+-----+----+--+
|0|1 4|2 5 8|3 6 9|7 10|11|
+-+---+-----+-----+----+--+
```

В общем случае, `u/.y` есть результат применения `u` к анти-диагоналям `_2`-ячеек `y` . Если ранг `y` меньше двух, вместо `y` берется матрица `,.y` .

`m/.y` применяет последовательные глаголы из герундия `m` к анти-диагоналям `_2`-ячеек `y`, продолжая `m` циклически как требуется. Тоесть:

```
   <`(<@|.) /. i.3 4
+-+---+-----+-----+----+--+
|0|4 1|2 5 8|9 6 3|7 10|11|
+-+---+-----+-----+----+--+
```

`x u/.y` ↔ `(=x) u@# y` , тоесть элементы `x` служат ключами для соответствующих элементов `y` , и `u` применяется к каждой группе элементов `y` , соответствующих одинаковым ключам. Например:

```
   1 2 3 1 3 2 1 </. 'abcdefg'
+---+--+--+
|adg|bf|ce|
+---+--+--+
```

`x m/.y` применяет последовательные глаголы из герундия `m` к группам элементов `y`, продолжая `m` циклически как требуется.

Применение функции к диагоналям матрицы требуется при вычислении корреляции, свертки и произведений коэффициентов многочленов (или, эквивалентно, для вычисления произведений чисел, записанных в некоторой системе счисления). Например:

```
   t=: p */ q [ p=: 1 2 1 [ q=: 1 3 3 1

   t ; (+//.t) ; 1 1 &(+//.@(*/)) ^: (i.6) 1
+-------+------------+------------+
|1 3 3 1|1 5 10 10 5 1|1 0  0  0 0 0|
|2 6 6 2|             |1 1  0  0 0 0|
|1 3 3 1|             |1 2  1  0 0 0|
|       |             |1 3  3  1 0 0|
|       |             |1 4  6  4 1 0|
|       |             |1 5 10 10 5 1|
```

```
+-------+-------------+-------------+

   ((10#.p)*10#.q), 10 #. +//. p */ q
161051 161051
```

В отличие от коэффициентов многочленов, диагональные суммы таблицы умножения цифр должны быть "нормализваны" если любая из них больше или равна основанию системы счисления.

По Возрастанию `/:` `_ _ _` Упорядочить

`/:` в применении к *любому* аргументу, дает вектор перестановки, такой что `(/:y){y` упорядочивает `y` по возрастанию. Например:

```
   n=: 3 1 4 2 1 3 3
   ]g=: /: n
1 4 3 0 5 6 2

   g { n
1 1 2 3 3 3 4
```

`x/:y` есть `(/:y){x` ; т.е. `x` упорядочивается по возрастанию `y`. В частности, `y/:y` (или `/:~y`) упорядочивает `y` . Например:

```
   y=: 'popfly'
   y /: 3 1 4 1 5 9
ofpply

   y /: y
floppy
```

Элементы `/:y` , выбирающие равные элементы `y` , расположены по возрастанию. Если `y` матрица, `/:y` сравнивает ее строки, считая их числами, записанными по основанию, большему чем удвоенный наибольший по модулю элемент матрицы. Элементы аргументов более высокого ранга перед сортировкой разбираются (`,.y`).

Если `y` текстовый, `/:y` производит сравнение в соответствии с лексикографическим порядком, определяемым алфавитом `a.` ; можно установить и другой порядок `cs` , упорядочивая по `cs i. y` . Например:

```
   ]n=: 3 1 4 1 6,2 7 1 8 3,:6 1 8 0 3
3 1 4 1 6
2 7 1 8 3
6 1 8 0 3

   /: n
1 0 2

   Aa=: ' ',. a. {~ 65 97 +/ i. 26
   x=: words=: >;: 'When eras die'
   j=: <./Aa i."1 _ x
   x ; (x/:x) ; (x/:j) ; Aa
+----+----+----+--------------------------+
|When|When|die | ABCDEFGHIJKLMNOPQRSTUVWXYZ|
|eras|die |eras| abcdefghijklmnopqrstuvwxyz|
|die |eras|When|                          |
+----+----+----+--------------------------+
```

Типы: числовой или пустой массив, символы, буквы (1-байтного или 2-х байтного набора символов), и упаковки упорядочиваются в указанном порядке; среди них, меньший ранг предшествует

большему, массивы при сравнении дополняются заполнителем до общей размерности. Комплексные аргументы сортируются по действительной, а затем мнимой части. Упаковки сортируются в соответствии с их распакованным содержимым.

Префиксно `m\ u\ _ 0 _` **В Окне**

u\y содержит #y элементов, получающихся применением u к каждому префиксу k{.y , для k от 1 до #y .

m\y применяет последовательные глаголы из герундия m к префиксам y , расширяя m циклически сколько требуется.

Если x>:0 , элементы x u\ y получаются применением u в скользящем окне длины x . Если x<0 , u применяется к *не перекрывающимся* окнам длины |x , включая остаток, если он есть.

x m\ y применяет последовательные глаголы из герундия m к окнам в y , расширяя m циклически сколько требуется.

```
   +/\a=: 1 2 4 8 16                    NB. Частичные суммы
1 3 7 15 31

   */\a                                 NB. Частичные произведения
1 2 8 64 1024

   <\a
+-+---+-----+-------+----------+
|1|1 2|1 2 4|1 2 4 8|1 2 4 8 16|
+-+---+-----+-------+----------+

   <\i.3 4
+-------+-------+---------+
|0 1 2 3|0 1 2 3|0 1  2  3|
|       |4 5 6 7|4 5  6  7|
|       |       |8 9 10 11|
+-------+-------+---------+

   (+/\^:_1 +/\ a) ,: */\^:_1 a
1 2 4 8 16
1 2 2 2  2
```

Следующие примеры иллюстрируют диаду *В Окне*:

```
   (2 -/\ ])  a                         NB. Обратные разности
_1 _2 _4 _8
   (2 -~/\ ]) a                         NB. Прямые разности
1 2 4 8

   3  <\ 'abcdefgh'
+---+---+---+---+---+---+
```

```
|abc|bcd|cde|def|efg|fgh|
+---+---+---+---+---+---+
   _3 <\ 'abcdefgh'
+---+---+--+
|abc|def|gh|
+---+---+--+
```

Суффиксно `m\.` `u\.` `_ 0 _` Вне Окна

`u\.y` содержит `#y` элементов, получающихся применением `u` к суффиксам `y` , начиная с суффикса длины `#y` (тоесть `y` целиком), и так далее до суффикса длины `1` .

`m\.y` применяет последовательные глаголы из герундия `m` к суффиксам `y` , расширяя `m` циклически сколько требуется.

Если `x>:0` в `x u\. y` , тогда `u` применяется вовне скользящего окна в `y` , тоесть, опуская элементы, принадлежащие скользящему окну длины `x` . Если `x<0` , подавляются элементы не перекрывающихся окон, последнее из которых может быть неполным.

`x m\.y` применяет последовательные глаголы из герундия `m` вовне окон в `y`, расширяя `m` циклически сколько требуется.

```
   */\. y=: 1 2 3 4 5
120 120 60 20 5

   <\. y
+---------+-------+-----+---+-+
|1 2 3 4 5|2 3 4 5|3 4 5|4 5|5|
+---------+-------+-----+---+-+

   3 <\. 'abcdefgh'
+-----+-----+-----+-----+-----+-----+
|defgh|aefgh|abfgh|abcgh|abcdh|abcde|
+-----+-----+-----+-----+-----+-----+

   _3 <\. 'abcdefgh'
+-----+-----+------+
|defgh|abcgh|abcdef|
+-----+-----+------+

   ]m=: i.3 3
0 1 2
3 4 5
6 7 8

   <"_2 (minors=: 1&(|:\.)"2^:2) m
+---+---+---+
|4 5|3 5|3 4|
|7 8|6 8|6 7|
+---+---+---+
|1 2|0 2|0 1|
```

```
|7 8|6 8|6 7|
+---+---+---+
|1 2|0 2|0 1|
|4 5|3 5|3 4|
+---+---+---+
```

По Убыванию `\:` `_ _ _` Упорядочить

`\:` в применении к *любому* аргументу дает вектор перестановки, такой что `(\:y){y` упорядочивает `y` по убыванию. Например:

```
   ]g=:\:y=:3 1 4 2 1 3 3
2 0 5 6 3 1 4

   g{y
4 3 3 3 2 1 1
```

`x\:y` есть `(\:y){x` ; т.е. `x` упорядочивается по убыванию `y` . В частности, `y\:y` (или `\:~y`) упорядочивает `y` .

Например:

```
   \:~ 'abecedarian'
rnieedcbaaa

   \:~"1 'dozen',:'disk'
zoned
skid
```

Элементы `\:y` , выбирающие равные элементы `y` , расположены по возрастанию. Если `y` матрица, `\:y` сравнивает значения ее строк, считая их числами, записанными по основанию, большему чем удвоенный наибольший по модулю элемент матрицы. Элементы аргументов более высокого ранга перед сортировкой разбираются (`,.y`).

Если `y` текстовый, `\:y` использует для сравнения лексикографический порядок, определяемый алфавитом `a.` ; можно установить и другой порядок `cs`, упорядочивая по `cs i. y` . Например:

```
   ]n=: 3 1 4 1 6,2 7 1 8 3,:6 1 8 0 3
3 1 4 1 6
2 7 1 8 3
6 1 8 0 3

   \: n
2 0 1

   \:~ >;:'when eras die, their legacies'
when
their
legacies
eras
die
,
```

См. По Возрастанию `(/:)` для описания работы с комплексными и упакованными аргументами.

Пропустить [ ]

Пропустить Левый, Правый

Монады [и] являются *тождественными* функциями; каждая просто возвращает свой аргумент.

x [y (*левая скобка*) возвращает левый аргумент x , и x] y (*правая скобка*) возвращает правый аргумент y .

Например:

```
   n=: i. 2 3
   a=: 'abcde'

   ]n
0 1 2
3 4 5

   [a
abcde

   n[a
0 1 2
3 4 5

   n]a
abcde

   ([\ ; ]\ ; [\. ; ]\.) 'ABCDEF'
+------+------+------+------+
|A     |A     |ABCDEF|ABCDEF|
|AB    |AB    |BCDEF |BCDEF |
|ABC   |ABC   |CDEF  |CDEF  |
|ABCD  |ABCD  |DEF   |DEF   |
|ABCDE |ABCDE |EF    |EF    |
|ABCDEF|ABCDEF|F     |F     |
+------+------+------+------+
```

[: отключает левую ветвь вилки, как описано в Разделе II F. Например, глагол p=: [: +/ + * - применяет монаду +/ к результату вилки + * - .

[: выдает ошибку при вызове с любым аргументом.

Шапки позволяют определить более широкий спектр функций в виде цепочек глаголов. Например: максимум, деленный на произведение суммы и разности, можно легко записать в виде одной цепочки; в то время как (без использования шапки), выражение: максимум, деленный на (монаду) пол произведения суммы и разности, потребовало бы использование цепочек, прерванных монадой. Тоесть:

```
   f=: >. % + * -
   g=: >. % <. @ (+ * -)

   2.5 f 4
_0.410256

   2.5 g 4
_0.4
```

Шапка позволяет записать непрерывную цепочку в виде:

```
   h=: >. % [: <. + * -

   2.5 h 4
_0.4
```

Поскольку область определения шапки пуста, ее можно использовать (с :) для определения функции у которой вызов монадного или диадного случая приводит к ошибке. Например:

```
   abs=: | : [:
   res=: [: : |

   res _4 0 5
|valence error: res
|       res _4 0 5

   abs _4 0 5
4 0 5

   3 res _4 0 5
```

```
2 0 2

   3 abs _4 0 5
|valence error: abs
|   3     abs _4 0 5
```

Прямое Произведение `{ 1 0 _` Выбрать

{y формирует прямое произведение всех атомов своего аргумента; размерность производимого упаковочного массива получается соединением размерностей распакованных элементов y , а его распакованные элементы имеют (одинаковую) размерность $y . Например:

```
   { 'ht';'ao';'gtw'
+---+---+---+
|hag|hat|haw|
+---+---+---+
|hog|hot|how|
+---+---+---+
+---+---+---+
|tag|tat|taw|
+---+---+---+
|tog|tot|tow|
+---+---+---+
```

При помощи { можно легко определить Декартово произведение (Cartesian product):

```
   CP=: {@(,&<)
   0 1 CP 2 3 4
+---+---+---+
|0 2|0 3|0 4|
+---+---+---+
|1 2|1 3|1 4|
+---+---+---+
```

Если x целое в диапазоне от -n=:#y до n-1 , то x{y выбирает элемент n|x из y . Тоесть:

```
   2 0 _1 _3 { 'abcdefg'
cage

   t=:3 4$'abcdefghijkl'
   1{t
efgh
```

В более общем случае, >x может быть (возможно упаковочным) списком, элементы которого являются массивами, указывающими на выборку вдоль последовательных измерений y .

Наконец, если любой из r=:>j{>x все еще упакован, выборка производится по индексам (вдоль соответствующего измерения), которые *отсутствуют* в >r .

Заметьте, что результатом последнего диадного примера, тоесть (<<<_1){m , является все *кроме* последнего элемента.

```
   t=: 3 4 $ 'abcdefghijkl'
   t; (1{t); (2 1{t); (1{"1 t); ((,1){"1 t); (2 1{"1 t)
+----+----+----+---+-+--+
|abcd|efgh|ijkl|bfj|b|cb|
|efgh|    |efgh|   |f|gf|
|ijkl|    |    |   |j|kj|
+----+----+----+---+-+--+

   t; (2 0{t); ((<2 0){t); ((2 0;1 3){t); ((<2 0;1 3){t)
+----+----+-+--+--+
```

```
|abcd|ijkl|i|ih|jl|
|efgh|abcd| |  |bd|
|ijkl|    | |  |  |
+----+----+-+--+--+
```

```
   (_1{m); (_1{"2 m); (_1{"1 m); (<<<_1){m=:i.2 3 4
+-----------+-----------+--------+---------+
|12 13 14 15| 8  9 10 11| 3  7 11|0 1  2  3|
|16 17 18 19|20 21 22 23|15 19 23|4 5  6  7|
|20 21 22 23|           |        |8 9 10 11|
+-----------+-----------+--------+---------+
```

Голова `{. _ 1 _` Взять

`{.y` выбирает первый элемент `y` (или элемент заполнителя, если элементы в `y` отсутствуют); тоесть `{.y` ↔ `0{1{.y` . Таким образом:

```
   a=: i. 3 2 3
   a;({.a);({."2 a);({."1 a)
+-------+-----+--------+-----+
| 0  1  2|0 1 2| 0  1  2| 0  3|
| 3  4  5|3 4 5| 6  7  8| 6  9|
|        |     |12 13 14|12 15|
| 6  7  8|     |        |     |
| 9 10 11|     |        |     |
|        |     |        |     |
|12 13 14|     |        |     |
|15 16 17|     |        |     |
+-------+-----+--------+-----+

   ]b=: ;/a
+-----+-------+--------+
|0 1 2|6  7  8|12 13 14|
|3 4 5|9 10 11|15 16 17|
+-----+-------+--------+

   {.&> b
 0  1  2
 6  7  8
12 13 14

   {. i.0 3
0 0 0
```

Если `x` атом, `x{.y` выбирает `|x` последовательных элементов `y` (начиная с начала, при `x` положительном; заканчивая в конце, при отрицательном).

В случае *нехватки* (когда число запрашиваемых элементов превышает количество элементов в массиве) добавляются дополнительные элементы, состоящие из *заполнителя* (нулей, если `y` числовой; `a:` , если упаковочный; пробелов, если текстовый; и `s: ' '` , если символьный). Атом заполнителя `f` можно *настроить* при помощи `{.!.f` .

В общем случае, если `y` не атом, `x` может быть списком не длиннее `$$y` элементов; а если `y` атом, вместо него берется `((#x)$1)$y` . Элемент `k` производит `(k{x){."(($$y)-k) y` ; бесконечность заменяется на величину размерности вдоль соответствующего измерения.

Следующие примеры иллюстрируют диаду *взять*:

```
   y=: i. 3 4
   y;(2{.y);(5{.y);(_5{.y);(_6{.'abcd');(2 _3{.y)
+---------+-------+---------+---------+------+-----+
|0 1  2  3|0 1 2 3|0 1  2  3|0 0  0  0|  abcd|1 2 3|
|4 5  6  7|4 5 6 7|4 5  6  7|0 0  0  0|      |5 6 7|
|8 9 10 11|       |8 9 10 11|0 1  2  3|      |     |
|         |       |0 0  0  0|4 5  6  7|      |     |
|         |       |0 0  0  0|8 9 10 11|      |     |
+---------+-------+---------+---------+------+-----+

   2 {."1 y
0 1
4 5
```

```
8 9

   _ 2 {. y
0 1
4 5
8 9

   6{.'ab';'cde';'fghi'
+--+---+----++++
|ab|cde|fghi||||
+--+---+----++++
```

Хвост { : _

{:y выбирает последний элемент y , или элемент заполнителей, если y не содержит элементов; тоесть {:y ↔ 0{_1{.y . Атом заполнителя f можно *настроить* при помощи {:!.f .

```
   ]y=: a.{~ (a.i.'A') + i.4 5
ABCDE
FGHIJ
KLMNO
PQRST

   f=: }: ; {:
   f y
+-----+-----+
|ABCDE|PQRST|
|FGHIJ|     |
|KLMNO|     |
+-----+-----+

   g=: }: ,.@; {:
   g y
+-----+
|ABCDE|
|FGHIJ|
|KLMNO|
+-----+
|PQRST|
+-----+

   h=: {. ,.@; }.
   h y
+-----+
|ABCDE|
+-----+
|FGHIJ|
|KLMNO|
|PQRST|
+-----+

   {:"1 y
EJOT
```

Пути `{::  _ 1 _` Достать

`{::y` упакован так же как и `y` , а все его элементы представляют собой пути к каждому листу (каждому не упаковочному под-массиву).	`x{::y` достает под-массив из `y` , в соответствии с путем `x` ; выборка на каждом уровне основывается на `{` и, кроме последнего уровня, ее результатом должен быть атом.

Пути и Достать моделируются следующим образом:

```
   cat  =: { @: (i.&.>) @: $
   mapp =: 4 : 'if. L. y do. (<"0 x,&.><"0 cat y) mapp&.> y else. >x end.'
   map  =: a:&mapp
   fetch=: >@({&>/)@(<"0@|.@[ , <@]) " 1 _
```

Следующие фразы иллюстрируют использование глаголов Пути и Достать:

```
   ] y=: 1 2 3;4 5;i.4 5
+-----+---+--------------+
|1 2 3|4 5| 0  1  2  3  4|
|     |   | 5  6  7  8  9|
|     |   |10 11 12 13 14|
|     |   |15 16 17 18 19|
+-----+---+--------------+
```

`(2;_1 _1){::y`	Число 19
`(_1;3 4) {::y`	Число 19
`{::y`	Пути к каждому распакованному массиву
`{::cat L: 0 y`	Пути к каждому распакованному скаляру

```
   ] t=: 5!:2 <'fetch'
```

Массив с интересной структурой

```
+-------------------------------------------------------+-+---+
|+---------------+-+-------------------------------+|"|1 _|
||+-+-+----------+|@|+-------------------+-+-------+|| |   |
|||>|@|+-------+-+|| ||+--------------+-+-+|,|+-+-+-+||| |   |
||| | ||+-+-+-+|/||| |||+-------+-+--+|@|[|| ||<|@|]|||| |   |
||| | |||{|&|>|| ||| ||||+-+-+-+|@||.|| | || |+-+-+-+||| |   |
||| | ||+-+-+-+| ||| |||||<|"|0|| |  || | || |       ||| |   |
||| | |+-------+-+|| ||||+-+-+-+| |  || | || |       ||| |   |
||+-+-+----------+| |||+-------+-+--+| | || |       ||| |   |
||                | ||+--------------+-+-+| |       ||| |   |
||                | |+-------------------+-+-------+|| |   |
|+---------------+-+-------------------------------+| |   |
+-------------------------------------------------------+-+---+
```

`(0;2;0;0;0){:: t`	Достать под-массив, соответствующий `<"0` in `t`
`(0;2;0;0;0;_1){:: t`	Выбрать `0` там
`t ,&< L: 0 1 {:: t`	Пометить каждый лист путем к нему
`< S: 0 t`	Упакованные листья `t`
`< S: 1 {:: t`	Упакованные пути в `t`
`t ,&< S: 0 1 {:: t`	Матрица из двух столбцов: листья и пути
`# 0: S: 0 t`	Количество листьев в `t`

Выбрав m} _ _ _ Заменяя

Если m число и z=: m} y , то $z равно $m , и равно размерности *элемента* y . Каждый атом j{z при этом равен j{(j{m){y . Например:

```
   y=: a.{~(a.i.'A')+i.4 5
   m=: 3 1 0 2 1
   y ; m ; m}y
+-----+---------+-----+
|ABCDE|3 1 0 2 1|PGCNJ|
|FGHIJ|         |     |
|KLMNO|         |     |
|PQRST|         |     |
+-----+---------+-----+
```

Если m не герундий, то x m} y формируется, заменяя x части y , выбираемые m&{ (выдается ошибка, если эта выборка требует использования заполнителя). Тоесть:

```
   y; '%*'(1 3;2 _1)} y
+-----+-----+
|ABCDE|ABCDE|
|FGHIJ|FGH%J|
|KLMNO|KLMN*|
|PQRST|PQRST|
+-----+-----+
```

$x должно быть суффиксом $m{y , и x имеет тот-же эффект, что и ($m{y)$,x . Тоесть:

```
   y; 'think' 1 2} y
+-----+-----+
|ABCDE|ABCDE|
|FGHIJ|think|
|KLMNO|think|
|PQRST|PQRST|
+-----+-----+
```

Если m герундий, то один из его элементов определяет индексный аргумент наречия } , а другие изменяют аргументы x и y :

```
x (v0`v1`v2)} y  &harr;  (x v0 y) (x v1 y)} (x v2 y)
(v0`v1`v2)} y    &harr;  (v1 y)} (v2 y)
(v1`v2)} y       &harr;  (v1 y)} (v2 y)
```

Например, следующие функции E1, E2, и E3 меняют местами две строки в матрице, умножают строку на константу и добавляют одну строку, умножив ее константу, к другой:

```
   E1=: <@] C. [
   E2=: f`g`[}
   E3=: F`g`[}
   f=: {:@] * {.@] { [
   F=: [: +/ (1:,{:@]) * (}:@] { []
   g=: {.@]
   M=: i. 4 5
   M;(M E1 1 3);(M E2 1 10);(M E3 1 3 10)
+-------------+-------------+-------------+------------------+
```

```
| 0  1  2  3  4| 0  1  2  3  4| 0  1  2  3  4|  0   1   2   3   4|
| 5  6  7  8  9|15 16 17 18 19|50 60 70 80 90|155 166 177 188 199|
|10 11 12 13 14|10 11 12 13 14|10 11 12 13 14| 10  11  12  13  14|
|15 16 17 18 19| 5  6  7  8  9|15 16 17 18 19| 15  16  17  18  19|
+--------------+--------------+--------------+-------------------+
```

Выбрав `u}` _ _ _ Заменяя

`u}` определено в терминах случая с существительным `m}`, только для вычисления требуемых индексов к аргументу (или аргументам) применяется глагол `u`.

Например:

```
   x=: 100 + i. 2 4
   u=: */@$@] | (5: * i.@$@[)
   y=: i. 3 2 4
   x ; y ; (x u y) ; (x u} y)
+--------------+----------+----------+--------------+
|100 101 102 103| 0  1  2  3| 0 5 10 15|100 105   2   3|
|104 105 106 107| 4  5  6  7|20 1  6 11|  4 101 106   7|
|               |           |          |               |
|               | 8  9 10 11|          |  8   9 102 107|
|               |12 13 14 15|          | 12  13  14 103|
|               |           |          |               |
|               |16 17 18 19|          | 16  17  18  19|
|               |20 21 22 23|          |104  21  22  23|
+---------------+----------+----------+---------------+
```

Позиции, выбираемые `x u} y` могут зависеть от любого или от обоих аргументов `x` и `y`, для наиболее распространенных случаев можно определить удобные наречия. Например:

```
   A=: @(i.@$@])
   u=: (<0 1)&|:
   x=: 'DIAG' [ y=: a. {~ (a. i. 'a') + i. 4 5
   x;y;(x u A y);(x u A} y)
+----+-----+---------+-----+
|DIAG|abcde|0 6 12 18|Dbcde|
|    |fghij|         |fIhij|
|    |klmno|         |klAno|
|    |pqrst|         |pqrGt|
+----+-----+---------+-----+
```

Так же см. `m}` для герундия.

Обезглавить `}.` `_ 1 _` Отбросить

`}.` отбрасывает первый элемент своего аргумента.

`x }. y` отбрасывает (не более) `|x` элементов из `y` ; с начала, если `x` положительно; с конца, если отрицательно.

В общем случае, если `y` не атом, `x` может быть списком не длиннее `r=:$$y`, каждый его `k`-й элемент дает `(k{x) }."(r-k) y`; если `y` атом, результатом является `(0=x)$y` .

```
   ]y=: a. {~ (a. i. 'A') + i. 4 5
ABCDE
FGHIJ
KLMNO
PQRST

   f=: }. ; {.
   f y
+-----+-----+
|FGHIJ|ABCDE|
|KLMNO|     |
|PQRST|     |
+-----+-----+

   g=: }. ,.@; {.
   g y
+-----+
|FGHIJ|
|KLMNO|
|PQRST|
+-----+
|ABCDE|
+-----+

   (2}.y) ; (_2}.y) ; (6}.y) ; ($ 6}.y) ; (}."1 y) ; (3}."1 y)
+-----+-----+-----+---+----+--+
|KLMNO|ABCDE|     |0 5|BCDE|DE|
|PQRST|FGHIJ|     |   |GHIJ|IJ|
|     |     |     |   |LMNO|NO|
|     |     |     |   |QRST|ST|
+-----+-----+-----+---+----+--+
```

Купировать }: _

}:y отбрасывает последний элемент y , эквивалентно _1 }. y . Тоесть:

```
   ]y=: a. {~ (a. i. 'A') + i. 4 5
ABCDE
FGHIJ
KLMNO
PQRST

   f=: }: ; {:
   f y
+-----+-----+
|ABCDE|PQRST|
|FGHIJ|     |
|KLMNO|     |
+-----+-----+

   g=: }: ,.@; {:
   g y
+-----+
|ABCDE|
|FGHIJ|
|KLMNO|
+-----+
|PQRST|
+-----+

   h=: {. ,.@; }.
   h y
+-----+
|ABCDE|
+-----+
|FGHIJ|
|KLMNO|
|PQRST|
+-----+

   }:"1 y
ABCD
FGHI
KLMN
PQRS
```

С Рангом `m"n`

Глагол `m"n` производит постоянный результат `m` для каждой ячейки, к которой он применяется. Используемый ранг есть `3 $&.|. n` . Например, если `n=:2`, то все три ранга равны `2 2 2`, и если `n=: 2 3`, они `3 2 3` . Отрицательный ранг является дополнительным: `m"(-r) y` эквивалентно `m"(0>.(#$y)-r)"_ y` .

Тоесть:

```
   v=: 2 3 5 7
   m=: i. 2 3
   m ; (m"0 v) ; (m"1 v); (m"1 m)
+-----+-----+-----+-----+
|0 1 2|0 1 2|0 1 2|0 1 2|
|3 4 5|3 4 5|3 4 5|3 4 5|
|     |     |     |     |
|     |0 1 2|     |0 1 2|
|     |3 4 5|     |3 4 5|
|     |     |     |     |
|     |0 1 2|     |     |
|     |3 4 5|     |     |
|     |     |     |     |
|     |0 1 2|     |     |
|     |3 4 5|     |     |
+-----+-----+-----+-----+

   v m" 1 2 m
0 1 2
3 4 5
```

Глаголы от `_9:` до `9:` являются *постоянными* глаголами, и эквивалентны глаголам от `_9"_` до `9"_` . Например:

```
   odds=: 1: + 2: * i.
   odds 5
1 3 5 7 9
```

С Рангом u"n

Глагол u"n применяет u к каждой ячейке ранга n . Полная форма ранга есть 3$&.|.n . Например, если n=:2 , все три ранга равны 2 2 2 , а если n=: 2 3, они 3 2 3 . Отрицательный ранг является дополнительным: u"(-r) y эквивалентно u"(0>.(#$y)-r)"_ y .

См. также Раздел II.A Существительные и Раздел II.B Глаголы.

```
   [] ; , ; ,"2) y=: a. {~ (a.i.'A') + i. 2 3 4
+----+------------------------+------------+
|ABCD|ABCDEFGHIJKLMNOPQRSTUVWX|ABCDEFGHIJKL|
|EFGH|                        |MNOPQRSTUVWX|
|IJKL|                        |            |
|    |                        |            |
|MNOP|                        |            |
|QRST|                        |            |
|UVWX|                        |            |
+----+------------------------+------------+

   (<"0 ; <"1 ; <"2 ; <"3 ,&< <"_1) y     NB. упаковка с рангами 0 1 2 3 _1
+---------+----------------+-----------+------+-----------+
|+-+-+-+-+|+----+----+----+|+----+----+|+----+|+----+----+|
||A|B|C|D|||ABCD|EFGH|IJKL|||ABCD|MNOP|||ABCD|||ABCD|MNOP||
|+-+-+-+-+|+----+----+----+||EFGH|QRST|||EFGH|||EFGH|QRST||
||E|F|G|H|||MNOP|QRST|UVWX|||IJKL|UVWX|||IJKL|||IJKL|UVWX||
|+-+-+-+-+|+----+----+----+|+----+----+||    ||+----+----+|
||I|J|K|L||                |            ||MNOP||           |
|+-+-+-+-+|                 |            ||QRST||           |
|         |                 |            ||UVWX||           |
|+-+-+-+-+|                 |            |+----+|           |
||M|N|O|P||                 |            |      |           |
|+-+-+-+-+|                 |            |      |           |
||Q|R|S|T||                 |            |      |           |
|+-+-+-+-+|                 |            |      |           |
||U|V|W|X||                 |            |      |           |
|+-+-+-+-+|                 |            |      |           |
+---------+----------------+-----------+------+-----------+

   ('*#' ,"0 1 ' abcde') ; (+/"2 i. 2 3 4) ; (+/"_1 i. 2 3 4)
+-------+-----------+-----------+
|* abcde|12 15 18 21|12 15 18 21|
|# abcde|48 51 54 57|48 51 54 57|
+-------+-----------+-----------+
```

Присвоить Ранг

`m"v  u"v  mv lv rv`

Глаголы `m"v` и `u"v` эквивалентны `m"r` и `u"r`, где `r` есть список рангов `v`. Результат можно проверить используя наречие *свойства* `b.` для запроса ранга.

Например:

```
   , b. 0
_ _ _

   %. b. 0
2 _ 2

   ravel=: , " %.

   ravel b. 0
2 _ 2

   ]y=: a. {~ (a. i. 'A') + i. 2 3 4
ABCD
EFGH
IJKL

MNOP
QRST
UVWX

   ,y
ABCDEFGHIJKLMNOPQRSTUVWX

   ravel y
ABCDEFGHIJKL
MNOPQRSTUVWX
```

Выполнить ". 1 _ _ В Числа

". y выполняет предложение y . Если результатом является существительное, оно выдается в качестве результата ". y ; если это глагол, наречие, союз, или результат отсутствует — выдается пустой вектор.

x". y преобразует текстовый массив y в числа. Результат имеет размерность (}:$y),(1=n)}.n где n -- максимельное количество чисел, встреченных в одной строке. x есть числовой скаляр, которым заменяются неверно указанные числа и дополняются короткие строки. Обычные правила записи числовых констант смягчаются следующим образом:

- знак "минус" можно указать как - или _
- запятые внутри чисел игнорируются
- 0 перед десятичной точкой может быть опущен
- перез числом может стоять знак + ; перед степенью в экспоненциальной записи может стоять знак +

Например:

```
   ". s=: '5 * a=: 3 + i. 6'
15 20 25 30 35 40

   a
3 4 5 6 7 8

   do=: ".
   do t=: '3 % 5'
0.6
   do |. t
1.66667
   $ do ''
0

   ]program=: 'a=: 2^3' ,: '5*a'
a=: 2^3
5*a

   do program
8 40
```

```
   do 'sum=: +/'

   sum 1 2 3 4
10

   s ; _999". s=: '1 2 3','-4 .5',:'bad 3,141'
+---------+--------------+
|1 2 3    |   1    2    3|
|-4 .5    |  _4  0.5 _999|
|bad 3,141|_999 3141 _999|
+---------+--------------+
```

В Текст ": _ 1 _ Форматировать

Монадный случай дает стандартное (как в интерактивной сессии) форматирование, выводящее минимум один пробел между столбцами. Например:

```
   ]text=: ": i. 2 5
0 1 2 3 4
5 6 7 8 9

   $ text
2 9

   3 + text
|domain error
|   3  +text

   '*#' ,. text
*0 1 2 3 4
#5 6 7 8 9

   ": 'abcd'
abcd

   $ ": ''
0
```

x":y производит текстовое представление y в соответствии с форматом x . Каждый элемент x есть комплексное число w j. d , управляющее представлением соответствующего столбца y следующим образом:

|w указывает ширину стобца; если этого места недостаточно, в нем печатаются звездочки. Если w равно нулю, место выделяется автоматически.

|d указывает число знаков после десятичной точки (включая ее, если d не равно нулю).

Знак "минус" помещается перед первой цифрой. Если w>:0 и d>:0, столбец выравнивается вправо. Иначе, (если w<0 или d<0), результат выводится в экспоненциальной записи (с одной цифрой перед десятичной точкой) и выравнивается вдоль левого края по десятичной точке.

Для комплексных y выводится только действительная часть. См. ниже для упакованных y .

```
   n ; 6j2 ": n=: % i. 2 4
+-------------------------+-----------------------+
|     _      1      0.5 0.333333|      _   1.00  0.50  0.33|
|0.25 0.2 0.166667 0.142857|  0.25  0.20  0.17  0.14|
+-------------------------+-----------------------+

   (7j2 ": -n) ; (3j2 ": n)
+---------------------------+-----------+
|     __  _1.00  _0.50  _0.33|  _*********|
```

```
|  _0.25  _0.20  _0.17  _0.14|************|
+---------------------------+------------+
```

```
   6j3 0j_6 ": 1r2 ^ 1 1000 *"1 i.5 2
 1.000  9.332636e_302
 0.250  8.128549e_904
 0.063  7.079811e_1506
 0.016  6.166381e_2108
 0.004  5.370801e_2710
```

Количество цифр после точки (для чисел с плавающей точкой) можно настроить (`!.`) и 9!:10 . Например:

```
   (": ; ":!.6 ; ":!.4 ; ":!.15) %7
+--------+--------+------+-----------------+
|0.142857|0.142857|0.1429|0.142857142857143|
+--------+--------+------+-----------------+
```

Для упакованного правого аргумента двух-элементный левый аргумент указывает выравнивание содержимого, причем 0, 1, и 2 кодируют верх/центр/низ, и лево/центр/право. 9!:16 и 9!:17 задают выравнивание по умолчанию. 9!:6 и 9!:7 позволяют указать символы для отображения упаковок.

```
   x=: 2 3 $ (2 #&.> 1+i.6) $&.> 'abcdef'
   (": x) ,. ' ' ,. (2 1 ": x)
+----+-----+------+ +----+-----+------+
|a   |bb   |ccc   | |    |     | ccc  |
|    |bb   |ccc   | |    | bb  | ccc  |
|    |     |ccc   | | a  | bb  | ccc  |
+----+-----+------+ +----+-----+------+
|dddd|eeeee|ffffff| |    |     |ffffff|
|dddd|eeeee|ffffff| |    |eeeee|ffffff|
|dddd|eeeee|ffffff| |dddd|eeeee|ffffff|
|dddd|eeeee|ffffff| |dddd|eeeee|ffffff|
|    |eeeee|ffffff| |dddd|eeeee|ffffff|
|    |     |ffffff| |dddd|eeeee|ffffff|
+----+-----+------+ +----+-----+------+
```

И/Или m`n  m`v u`n  u`v

В английском, *герундий* есть существительное с окончанием -ing, имеющее, одновременно, свойства глагола (как, например, *cooking* в выражении *the art of cooking*). Союз И/Или производит герундий из двух глаголов. Герундий обычно используется в сочетании с Между (/) и Сообразно (@.):

```
   ]g=: +`*
+-+-+
|+|*|
+-+-+

   (g/1 2 3 4 5) ; (1+2*3+4*5)
+--+--+
|47|47|
+--+--+
```

В общем случае, И/Или производит герундий следующим образом: u`v есть au,av , где au и av являются (упакованными существительными) *атомными представлениями* (5!:1) глаголов u и v . Более того, m`n есть m,n и m`v есть m,av , а u`n есть au,n . См. Bernecky and Hui [12]. Герундий можно произвести и непосредственно при помощи упаковки. Тоесть:

```
   ]h=: '+' ; '*'
+-+-+
|+|*|
+-+-+

   h/1 2 3 4 5
47
```

Атомное представление существительного (обозначенное так, чтобы отличить, например, существительное '+' от глагола +) производится следующей функцией:

```
   (ar=: [: < (,'0')"_ ; ]) '+'
+-----+
|+-+-+|
||0|+||
|+-+-+|
+-----+

   *`(ar '+')
+-+-----+
|*|+-+-+|
```

```
| ||0|+||
| |+-+-+|
+-+-----+
```

Выполнив (Герундий) `m`:n` — — —

Этот союз определен для трех случаев:

`m `: 0`	*Присоединяя*	Соединяет результаты индивидуальных глаголов.
`m `: 3`	*Между*	Вставляет глаголы между элементами. Эквивалентно `m/`
`m `: 6`	*Цепочкой*	Производит цепочку из глаголов герундия.

Например:

```
   <+:`-:`% `: 0 a=: 1 2 3 4 5
+-------------------------+
|  2   4        6    8  10|
|0.5   1      1.5    2 2.5|
|  1 0.5 0.333333 0.25 0.2|
+-------------------------+

   (+ b.0) ; (%. b.0) ; (+`%.`:0 b.0)
+-----+-----+-----+
|0 0 0|2 _ 2|_ _ _|
+-----+-----+-----+

   (+`* `:3 a) ; (+`*/a) ; (1+2*3+4*5)
+--+--+--+
|47|47|47|
+--+--+--+

   (+`*`- `: 6 a) ; ((+ * -) a)
+----------------+----------------+
|_1 _4 _9 _16 _25|_1 _4 _9 _16 _25|
+----------------+----------------+
```

Поверх `u@v  mv lv rv`

`u@v  y` ↔ `u  v  y` . Например, `+:@-  7` дает `_14` (удвоить взятое с обратным знаком). Более того, монадные случаи u@v и u&v эквивалентны.	`x  u@v  y` ↔ `u  x  v  y` . Например, `3  +:@-  7` равно `_8` (удвоенная разность).

Поскольку наречия и союзы (более точно определено в Разделе II E) выполняются перед глаголами, фразы с ними обычно используются в цепочках без скобок. Например:

```
   mean=: +/ % #
   mean 1 2 3 4
2.5

   f=: +:@*: +/ -:@%:     NB. Таблица сложения удвоенного квадрата и половины кор
   f 1 2 3 4
 2.5 2.70711 2.86603  3
 8.5 8.70711 8.86603  9
18.5 18.7071  18.866 19
32.5 32.7071  32.866 33
```

Поскольку союз применяется к обьекту, стоящему непосредственно справа, выражения справа от союзов обычно требуют использования скобок. Например:

```
   g=: *:@(+/)
   h=: *:@+/
   g 1 2 3 4
100
   h 1 2 3 4
6770404

   k=: *:@+
   k/ 1 2 3 4
6770404
```

Сравните поведение `@` и `@:` (они отличаются только рангом производимых ими глаголов).

Сообразно `m@.n  m@.v  mv lv rv`

`m@.n` есть глагол, определенный герундием `m` *сообразно* `n` ; тоесть цепочка глаголов, выбранная из `m` , по индексам в `n` . Если `n` упаковочный, в цепочку вставляются скобки. Случай `m@.v` берет в качестве индексов результат глагола `v` .

Например:

```
   dorh=: +: ` -: @. (]>9:)
   dorh " 0 primes=: 2 3 5 7 11 13 17 19
4 6 10 14 5.5 6.5 8.5 9.5

   _:`%:`*: @. * "0 a=: 2 1 0 _1 _2
1.41421 1 _ 1 4

   g=: +`-`*
   x=: 1 2 3 [ y=: 6 5 4
   (x g@.2: y)
6 10 12

   (] * <:) y=: 5 4 3 2 1 0
20 12 6 2 0 0

   1:`(] * <:) @. (1: < ]) "0 y
20 12 6 2 1 1

   1:`(] * $:@<:)@.(1: < ])"0 y
120 24 6 2 1 1

   +`-`*`% @. (1 0 3;2 0)
(- + %) (* +)

   3 +`-`*`% @. (1 0 3;2 0) 4
_12.8125

   perm=: i.@i.@_2: ` ([: ,/ (0: ,. $:&.<:) {"2 1 \:"1@=@i.) @. *
   perm 3
0 1 2
0 2 1
1 0 2
1 2 0
2 0 1
2 1 0
```

Удвоить или уполовинить (оператор Cas

Множители факториала

оператор Case

Вызвать Себя для рекурсии

Над `u@:v _ _ _`

`@:` эквивалентно `@` , кроме того, что ранги бесконечны.

Например:

```
   x=: 1 2 3 4
   y=: 7 5 3 2
   x */ @: + y
2016
```

Применяет произведение над (целым списком

```
   x */ @ + y
8 7 6 6
```

Применяет произведение поверх (каждой) сум

```
   + b. 0
0 0 0

   */ @: + b. 0
_ _ _

   */ @ + b. 0
0 0 0
```

C m&v u&n _
0 _

m&v y определено как m v y ; тоесть левый аргумент m цепляется к диаде v производя монаду (получается диада C фиксированным аргументом, тоесть монада). Подобно, u&n y определено как y u n ; другими словами, аргумент n цепляется к диаде u справа, производя монаду.

x m&v y ↔ m&v^:x y
x u&n y ↔ u&n^:x y

Например:

```
   10&^. 2 3 10 100 200
0.30103 0.477121 1 2 2.30103

   base10log=: 10&^.
   base10log 2 3 10 100 200
0.30103 0.477121 1 2 2.30103

   sine=: 1&o.
   sine o. 0 0.25 0.5 1.5 2
0 0.707107 1 _1 0

   ^&3 (1 2 3 4 5)
1 8 27 64 125

   ^&2 3"0 (1 2 3 4 5)
 1   1
 4   8
 9  27
16  64
25 125
```

Использование C часто называют *Currying* в честь Haskell-a Curry.

Фраза x f@[&0 y эквивалентна f^:x y , и применяет монаду f x раз к y . Например:

```
   fib=: (0 1,:1 1)&(+/ .*)@[&0 & 0 1
   fib i.10
 0  1
 1  1
 1  2
```

```
 2  3
 3  5
 5  8
 8 13
13 21
21 34
34 55
```

3a u&v mv mv mv

u&v y ↔ u v y . Тоесть +:&- 7 дает _14 (удвоить взятое с обратным знаком). Более того, монады u&v и u@v эквивалентны.	x u&v y ↔ (v x) u (v y) . Например, 3 +&! 4 есть 30, сумма факториалов.

Монадный случай эквивалентен композиции функций, используемой в математике, но диадный открывает новые возможности. Например:

```
   3 +&^. 4                    NB. Сумма натуральных логарифмов
2.48491

   ^ 3 +&^. 4                  NB. Умножение, используя натуральные логарифмы
12

   3 +&(10&^.) 4               NB. Сумма логарифмов по основанию 10
1.07918

   10 ^ 3 +&(10&^.) 4          NB. Умножение, используя логарифмы по осн. 10
12

   3 +&.^. 4                   NB. См. союз Преобразуя (&.)
12

   3 +&.(10&^.) 4
12
```

Сравните поведение & с поведением &: (они отличаются только рангом производимых ими глаголов).

Преобразуя (Дуально)

u&.v mv mv mv

Глагол u &.v эквивалентен композиции u & v кроме того, что к каждой ячейке результата применяется обращение глагола v . Тоесть (с монадным рангом v),

```
    u&.v y &harr; vi u v y
  x u&.v y &harr; vi (v x) u (v y)
```

где vi есть обращение v . Обращение обычно является обратным глаголом, как обсуждается более детально в описании союза в степени ^: .

```
   3 +&.^. 4                    Обращение натурального логарима есть экспо
12

   (^.^:_1) (^.3)+(^.4)
12

   (<b), <|. b=: 1 2 3 ; 2 3 5 7 ; 'abcde'
+---------------------+---------------------+
|+-----+-------+-----+|+-----+-------+-----+|
||1 2 3|2 3 5 7|abcde|||abcde|2 3 5 7|1 2 3||
|+-----+-------+-----+|+-----+-------+-----+|
+---------------------+---------------------+

   each=: &.>                   Наречие
   (<|. &. > b),(<|. each b)    Распаковывая
+---------------------+---------------------+
|+-----+-------+-----+|+-----+-------+-----+|
||3 2 1|7 5 3 2|edcba|||3 2 1|7 5 3 2|edcba||
|+-----+-------+-----+|+-----+-------+-----+|
+---------------------+---------------------+
```

В математике некоторые случаи функций под *преобразованием* называются *дуальными* или, *дуальными по отношению к*:

```
   f=: +. &. -.                 Дуально по отношению к Не
   f/~ d=: 0 1
0 0
0 1

   D=: &. -.                    Наречие "дуально по отношению к Не"
   (+.D/~d);(*./~d);(=D/~d);(~:/~d)
+---+---+---+---+
```

```
|0 0|0 0|0 1|0 1|
|0 1|0 1|1 0|1 0|
+---+---+---+---+
```

```
   DWL=: &.^.                    Дуально к логарифму
   DAN=: &. -                    Дуально к перемене знака
   (3 + DWL 4),(3*4),(3 <. DAN 4) , (3 >. 4)
12 12 4 4
```

Преобразуя (Дуально)

u&.:v _ _ _

```
u &.: v &harr; u&.(v"_)
```

```
   +/&.:^. 2 3 4
24
   +/&.^. 2 3 4
2 3 4
```

После u&:v _ _ _

&: эквивалентно & , кроме того, что ранги производимого глагола бесконечны; это соотношение подобно соотношению между @: и @ .

Например:

```
   a=: 'abcd' ; 'efgh'
   b=: 'ABCD' ; 'EFGH'

   a
+----+----+
|abcd|efgh|
+----+----+

   b
+----+----+
|ABCD|EFGH|
+----+----+

   a ,&:> b
abcd
efgh
ABCD
EFGH

   a ,&> b
abcdABCD
efghEFGH

   > b. 0
0 0 0

   , & > b. 0
0 0 0

   , &: > b. 0
_ _ _
```

Бросить Кубик

`? 0 0 0`

Сдать Карты

`? y` выбирает однородно распределенный случайный элемент из `i.y` если `y` положительно, или из интервала чисел больше 0 и меньше 1, если `y` равно 0. Начальная *затравка* генератора случайных чисел есть 7^5 (16807).

`x ? y` есть список `x` элементов, выбранных случайно *без повторений* из `i.y` .

```
   ? 6
0

   ? 6 6 6 6 6 6 6
0 5 5 4 2 3 2 1

   6 ? 6                    NB. Случайная перестановка
0 1 5 4 3 2

   mean=: +/ % #
   mean ? 1000 # 6
2.459

   ] m=: ? 4 4 $ 9          NB. Случайная матрица для экспериментирования
3 8 8 4
2 0 2 7
4 2 3 5
1 1 7 2

   -/ . * m                  NB. Определитель m
_659

   f=: ?@$ % ] - 1:
   3 6 f 9                   NB. Случ. таблица 3x6 из [0,1] с разрешением 9
0.375     1     1  0.5  0.25     0
 0.25 0.875   0.5 0.25 0.375 0.625
0.125 0.125 0.875 0.25 0.875  0.75

   ? 2 1 $ !38x
202466594106002578660243928542852207957556146
 796051165631186210909260717820924151231817l3
```

Для управления работой случайных функций определены несколько внешних глаголов. `9!:(2*n)` позволяет запросить параметр, а `9!:(1+2*n)` устанавливает его.

9!:42 y **Выбор ГСЧ.** Выбирает генератор случайных чисел

9!:43 y

y	ГСЧ	верхний предел	период	период
1	GB_Flip	2^31	_1+2^55	3.60e16
2	Mersenne Twister	2^32	_1+2^19937	4.32e6001
3	DX-1597-4d	_1+2^31	_1+(_1+2^31)^1597	1.24e14903
4	MRG32k3a	_209+2^32	-:*/	3.14e57
0	sum of above RNGs			

"Скорость" -- время, требующееся на генерацию одного миллиона случайных чисел из множества U(0,1); быстрейшему генератору соответствует время 1.0. По умолчанию ? вызывает Mersenne Twister, в качестве фиксированного ГСЧ ?. использует GB_Flip.

9!:44 y 9!:45 y **Состояние ГСЧ.** Последовательность генерируемых случайных чисел полностью зависит от состояния ГСЧ. Состояние представлено упаковочным вектором, интерпретация которого зависит от ГСЧ. Например:

```
   t=: 9!:44 ''  NB. запросить состояние ГСЧ
   5 ?@$ 10000
8590 6147 5158 4729 3522
   2 4 ?@$ 10
2 5 5 5
3 8 0 0
   9!:45 t       NB. установить состояние ГСЧ

   5 ?@$ 10000
8590 6147 5158 4729 3522
```

9!:0 y 9!:1 y **Затравка ГСЧ.** Запрашивает и устанавливают затравку, используемую для генерации случайных чисел. Начальное значение 7^5 . Затравка может представлять собой целый скаляр или (для Mersenne Twister) список целых.

128!:4 y **Прямой Доступ к ГСЧ.** Производит y целых, в том виде, в котором они непосредственно генерируются (обычно используется для отладки ГСЧ).

Алфавит/Ас `a.` `a:`

`a.` есть список элементов алфавита; он определяет *лексикографический порядок*, используемый при упорядочении (`/:` и `\:`) строк. Для разных вычислительных машин, содержимое `a.` и порядок его элементов могут быть различными.

В алфавите `$a.` элементов, просмотреть их в виде 32-колоночной таблицы можно при помощи `8 32$a.` . Присутствие некоторых *управляющих* (таких как возврат каретки) и *не-печатных* символов портят отображение этой таблицы. Собственно алфавит можно получить при помощи:

```
   1 2 3 { 8 32 $ a.
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

Возврат каретки обычно имеет индекс 13 (это можно проверить введя `13{a.`), индексы пробела и других символов можно определить следующим образом:

```
   a. i. 'aA +-*%'
97 65 32 43 45 42 37
```

Ас (единица, от латинского *as*) обозначается `a:` . Это упакованный пустой список `<$0` .

Индекс Перестановки `A. 1 0 _` Переставить

Если `T` таблица всех `!n` перестановок порядка `n` , упорядоченных в лексикографическом порядке (т.е., `/:T` есть `i.!n`), тогда `k` называется *индексом* перестановки `k{T` .

`A.` в применении к циклической или прямой перестановке дает ее индекс: `A. 0 3 2 1` есть `5`, так же как `A. 3 2 1` , `A.<3 1` и `A.0;2;3 1` .

Выражение `k A. b` переставляет элементы `b` при помощи перестановки порядка `#b` с индексом `k` .

Например:

```
   (A. 0 3 2 1) , (A. <3 1)
5 5

   A. |. i.45
119622220865480194561963161495657715064383733759999999999

   <: ! 45x
119622220865480194561963161495657715064383733759999999999

   tap=: i.@! A. i.           NB. Таблица всех перестановок

   (tap 3);(/: tap 3);({/\ tap 3);(/:{/\ tap 3)
+-----+-----------+-----+-----------+
|0 1 2|0 1 2 3 4 5|0 1 2|0 1 5 2 4 3|
|0 2 1|           |0 2 1|           |
|1 0 2|           |1 2 0|           |
|1 2 0|           |2 0 1|           |
|2 0 1|           |1 2 0|           |
|2 1 0|           |1 0 2|           |
+-----+-----------+-----+-----------+
```

В частности, `1 A. b` меняет местами два последних элемента `b`, , `_1 A. b` переставляет их в обратном порядке `3 A. b` , а `4 A. b` циклически проворачивает последние три элемента `b`. Например:

```
   b=: 'ABCD'

   (0 3 2 1{b);(0 3 2 1 C.b);((<3 1)C.b);(3 4 A.b)
+----+----+----+----+
|ADCB|ADCB|ADCB|ACDB|
```

```
|    |    |    |ADBC|
+----+----+----+----+

   (_19 5 A. b) ; (_19 |~ ! # b)
+----+-+
|ADCB|5|
|ADCB| |
+----+-+
```

Поразрядно `m b. _ 0 0`

Монада эквивалентна случаю нулевого левого аргумента; тоесть
`m b. y` ↔ `0 m b. y`

Если `f` диадная булевская функция, а `d=: 0 1 , тогда d f/ d` (или `f/~d`) есть ее полная таблица истинности. Например, ниже приведены таблицы для *или*, *не-или*, *и*, и *не-и*:

```
   (+./~ ; +:/~ ; *./~ ; *:/~) d=: 0 1
+---+---+---+---+
|0 1|1 0|0 0|1 1|
|1 1|0 0|0 1|1 0|
+---+---+---+---+
```

Если все эти таблицы отсортировать, то каждая из шестнадцати возможных булевских диад получит некоторый порядковый номер `k` ; фраза `k b.` тогда реализует соответствующую функцию. Кроме того, допускаются отрицательные индексы и аргументы в виде массивов.

Атом `m=16+k` обозначает побитовую булевскую функцию целых аргументов. Каждый атом ее аргументов считается битовыми списком длины `w` , где `w` -- длина слова машины, на которой работает программа. Например, `17 b.` выполняет побитовое *и*.

Наконец, `32 b.` соответствует *циклическому сдвигу*, `33 b.` *сдвигу*, а `34 b.` *сдвигу с учетом знака*.

В следующей таблице перечислены все возможные булевские функции:

m	Разобранная Таблица	Функция
`0 _16`	`0 0 0 0`	`0`
`16`	`1 1 1 1`	`1`

```
1  _15 17    0 0 0 1          x *. y
2  _14 18    0 0 1 0          x > y
3  _13 19    0 0 1 1          x
4  _12 20    0 1 0 0          x < y
5  _11 21    0 1 0 1          y
6  _10 22    0 1 1 0          x ~: y
7  _9  23    0 1 1 1          x +. y
8  _8  24    1 0 0 0          x +: y
9  _7  25    1 0 0 1          x = y
10 _6  26    1 0 1 0          -. y
11 _5  27    1 0 1 1          x >: y
12 _4  28    1 1 0 0          -. x
13 _3  29    1 1 0 1          x <: y
14 _2  30    1 1 1 0          x *: y
15 _1  31    1 1 1 1          1
       32                     циклический сдвиг
       33                     сдвиг
       34                     сдвиг с учетом знака
```

Например:

```
   (7 b./~ ; 8 b./~ ; 1 b./~ ; 14 b./~) d=: 0 1
+---+---+---+---+
|0 1|1 0|0 0|1 1|
|1 1|0 0|0 1|1 0|
+---+---+---+---+

   (_1 b./~ ; _3 b./~ ; _15 b./~) d       NB. отрицательные индексы
+---+---+---+
|1 1|1 1|0 0|
|1 1|0 1|0 1|
+---+---+---+

   (<"2) 2 0 1 |: 7 8 1 15 b./~ d         NB. аргументы в виде массивов
+---+---+---+---+
|0 1|1 0|0 0|1 1|
|1 1|0 0|0 1|1 1|
+---+---+---+---+

   12345 (17 b.) 67890                    NB. побитовое и
48
   f=: (32#2)&#: { '.x'"_
   f 12345 67890 48
..................xx......xxx..x
..............x....x..x..xx..x.
..........................xx....

   _12345 (23 b.) 67890                   NB. побитовое или
_12297
```

```
   f _12345 67890 _12297
xxxxxxxxxxxxxxxxxx..xxxxxx...xxx
..............x....x..x..xx..x.
xxxxxxxxxxxxxxxxxx..xxxxxxxx.xxx

   20 b./~ i.10                          NB. таблица побитового меньше
0 1 2 3 4 5 6 7 8 9
0 0 2 2 4 4 6 6 8 8
0 1 0 1 4 5 4 5 8 9
0 0 0 0 4 4 4 4 8 8
0 1 2 3 0 1 2 3 8 9
0 0 2 2 0 0 2 2 8 8
0 1 0 1 0 1 0 1 8 9
0 0 0 0 0 0 0 0 8 8
0 1 2 3 4 5 6 7 0 1
0 0 2 2 4 4 6 6 0 0

   23 b./\ 2^i.10                        NB. побитовое или с накоплением
1 3 7 15 31 63 127 255 511 1023

   _5 (33 b.) 12345                      NB. сдвиг
385
   f 12345 385
..................xx......xxx..x
......................xx......x

   _5 (33 b.) _12345                     NB. сдвиг
134217342
   f _12345 134217342
xxxxxxxxxxxxxxxxxx..xxxxxx...xxx
.....xxxxxxxxxxxxxxxxx..xxxxxx.

   _5 (34 b.) _12345                     NB. сдвиг с учетом знака
_386
   f _12345 _386
xxxxxxxxxxxxxxxxxx..xxxxxx...xxx
xxxxxxxxxxxxxxxxxxxxxx..xxxxxx.
```

Основные Свойства

u b. _

u b. y дает линейное представление обращения u , если y равно _1; ранги, если y равно 0; линейное представление тождественной функции, если y равно 1 .

Например:

```
   ^ b. _1
^.

   ^ b. 0
0 0 0

   ^ b. 1
$&1@(}.@$)

   g=: +&2@(*&3@*:)
   ]y=: g 5
77

   g ^:_1 y
5

   g b. _1
%:@(%&3)@(-&2)

   %:@(%&3)@(-&2) y
5

   g b. 0
0 0 0
```

Циклы C. 1 1 _ Разместить

Если p -- перестановка атомов i.n, то p называется вектором перестановки порядка n; а если n=#b, то p{b переставляет элементы b соответственно.

C.p дает список упакованных списков атомов i.#p, называемую *стандартным* циклическим представлением перестановки p . Тоесть, если p=:4 5 2 1 0 3, то C.p есть (,2);4 0;5 3 1 , поскольку перестановка p перемещает в позицию 2 элемент 2, в 4 элемент 0, в 0 элемент 4, в 5 элемент 3, в 3 элемент 1, и в 1 элемент 5. Монада C. обратна сама себе: в применении к стандартному циклическому представлению она дает соответствующий вектор перестановки.

Представление данной перестановки циклами неоднозначно; стандартная форма единственна в силу следующих дополнительных ограничений: циклы не пересекаются и полны (т.е., атомы упакованных элементов вместе представляют собой вектор перестановки); каждый упакованный цикл поворачивается так, чтобы он начинался со своего наибольшего элемента; и упакованные циклы располагаются по возрастанию своих первых элементов.

C. расширено до неотрицательных нестандартных

Если p и c являются стандартным и циклическим представлением перестановки порядка #b, то p C. b и c C. b размещают элементы b соответственно. Аргументы p и c могут быть определены *нестандартным* образом. В частности, можно использовать отрицательные целые вплоть до -#b , интерпретируемые как остатки от их деления на #b .

Если q не упакован, а элементы (#b)|q все разные, то q C. b эквивалентно p{b , где p есть стандартная форма q , определяемая как p=:((i.n)-.n|q),n|q , для n=:#b . Другими словами, позиции, перечисленные в q , перемещаются в конец.

Если q упаковка, элементы (#b)|>j{q должны быть разными для всех j , и упаковки применяются последовательно. Например:

```
   (2 1;3 0 1) C. i.5
1 2 3 0 4

   (<2 1) C. (<3 0 1) C. i.5
1 2 3 0 4

   q=: C. p=: 1 2 3 0 4 [ a=: 'abcde'
   q ; (q C. a) ; (p C. a) ; (p { a)
+-----------+-----+-----+-----+
|+-------+-+|bcdae|bcdae|bcdae|
||3 0 1 2|4||     |     |     |
|+-------+-+|     |     |     |
+-----------+-----+-----+-----+
```

случаев, при этом аргумент q считается представлением перестановки порядка 1+>./; q .

Монада C.!.2 вычисляет *четность* перестановки p , равную 1 или _1 в зависимости от того -- четно или нечетно число парных перестановок, требуемое для получения p из тождественной перестановки i.#p (и 0 если p не перестановка). Например:

```
   ] x=: 2 , (i.4) ,: 1 0 2 3
2 2 2 2
0 1 2 3
1 0 2 3
   C.!.2 x
0 1 _1
```

```
   a ; (<0 _1) C. a
+-----+-----+
|abcde|ebcda|
+-----+-----+
```

Дополнительные примеры:

```
   ] p=: 22 ?. 22               NB. Случайная перестановка порядка 22
16 18 21 8 6 15 10 14 7 11 0 2 5 3 9 12 20 17 4 19 13 1

   C. p                         NB. Ее циклы
+-------+--+--+----------------------------------------+
|15 12 5|17|19|21 1 18 4 6 10 0 16 20 13 3 8 7 14 9 11 2|
+-------+--+--+----------------------------------------+

   */ #&> C. p          NB. НОК длин циклов
51

   # ~. p&{^:(i.200) i.#p   NB. Размер подгруппы, генерируемой p
51
```

Глагол CT вычисляет *тензор Леви-Чивита* (абсолютно анти-симметричный тензор) порядка n в виде разреженного массива; элемент (<i){CT n есть четность перестановки индексов i .

```
   CT=: 3 : '(C.!.2 p) (<"1 p=. (i.!y) A. i.y)}1$.$~y'

   CT 3
0 1 2 |  1
0 2 1 | _1
1 0 2 | _1
1 2 0 |  1
2 0 1 |  1
2 1 0 | _1
```

```
   ($.^:_1 CT 3) ; ,"2 ' ' ,"1 '012'{~ >{ i.&.> $~3
+--------+------------+
| 0  0  0| 000 001 002|
| 0  0  1| 010 011 012|
| 0 _1  0| 020 021 022|
|        |            |
| 0  0 _1| 100 101 102|
| 0  0  0| 110 111 112|
| 1  0  0| 120 121 122|
|        |            |
| 0  1  0| 200 201 202|
|_1  0  0| 210 211 212|
| 0  0  0| 220 221 222|
+--------+------------+

   (CT 3) -: C.!.2&> { i.&.> $~ 3
1

   ] m=: ?. 3 3$10
6 5 9
2 4 9
0 7 0

   +/ , (CT #m) * *// m
_252
   -/ .* m
_252
```

NB. Детерминант

Производная u d. n 0

u d. n работает как u D. n , кроме того, что u считается функцией ранга 0.

```
   *: d. 1
+:

   *: d. 2
2"0

   (1: + _3&* + *:) d. 1
_3 2&p.

   (1: + _3&* + *:) d. 2
2"0

   (1: + _3&* + *:) d. 3
0"0

   ^. d. 1
%

   (^. * *:) d. 1
(% * *:) + ^. * +:

   (^. % *:) d. 1
((% * *:) - ^. * +:) % 0 0 0 0 1&p.

   (^. @ *:) d. 1
+: * %@*:

   *: d. _1
0 0 0 0.33333333333333331&p.

   % d. _1
^.

   *: d. 1 x=: _3 _2 _1 0 1 2 3
_6 _4 _2 0 2 4 6
   +: x
_6 _4 _2 0 2 4
```

Производная `m D. n  u D. n  mu`

`u D. n` есть n-ная производная `u`, и `u`v D. n` есть `u` с присвоенной n-ной производной `v` . Например:

```
   (cube D.1;cube D.2; (cube=: ^&3"0) D.3)y=: 2 3 4
+--------+--------+-----+
|12 27 48|12 18 24|6 6 6|
+--------+--------+-----+
```

Производная работает для постоянных функций, многочленов, экспоненты `^`, целых степеней `^&n`, и функций, производная которых присвоена фразой `u`v D. n` . Она так же работает для функций, полученных из перечисленных сложением, умножением и делением (`u+v`, и т.д.); вложенных функций `u@v`; и обратных функций `u^:_1` . Поскольку многие другие функции, такие как `j.` , `-` (перемена знака), `%:` (квадратный корень), `1&o.` (sin) и `6&o.` (cosh) сводятся к вышеперечисленным, они тоже могут быть автоматически дифференцированы. Остальные функции дифференцируются численно. Производная произвольной функции может быть рассмотрена как приближение ее многочленом (осуществленная при помощи деления матриц) или как наклон секущей `D:` .

Если ранг `u` равен `a` и ранг его результата равен `r`, то ранг аргумента `u D.1` тоже `a`, но ранг его результата равен `r+a` : результат `u D.1` представляет собой производную каждого атома результата `u` по отношению к каждому элементу его аргумента, в математике это называется *частные производные*. Например:

```
   volume=: */"1
   VOLUMES=: */\"1
   (volume;volume D.1;VOLUMES;VOLUMES D.1) y
+--+------+------+------+
|24|12 8 6|2 6 24|1 3 12|
|  |      |      |0 2  8|
|  |      |      |0 0  6|
+--+------+------+------+

   determinant=: -/ . *
   permanent=: +/ . *
   (];(determinant D.1);(permanent D.1))m=:*:i.3 3
+--------+-------------+-------------+
| 0  1  4|_201  324 _135|2249 1476 1017|
| 9 16 25| 132 _144   36| 260  144   36|
|36 49 64| _39   36   _9|  89   36    9|
+--------+-------------+-------------+
```

Наречия `D=: 1 : 'u"0 D.1'` и `VD=: 1 : 'u"1 D.1'` присваивают ранги своим аргументам, а потом берут первые производные; их удобно использовать в скалярном и векторном математическом анализе:

```
   sin=: 1&o.
   x=: 0.5p1 _0.25p1
   (*/\ VD y);(sin x);(sin D x);(sin D D x)
+------+----------+----------+----------+
|1 3 12|1 _0.707107|0 0.707107|_1 0.707107|
|0 2  8|           |          |           |
|0 0  6|           |          |           |
+------+----------+----------+----------+
```

u D: n mu mu

Секущая (наклон)

x u D: 1 y есть *наклон секущей* функции u , проходящей через точки y и y+x . Наклон секущей обобщен и на случай x u D. n y тем же образом, что и производная D. . Аргумент x может быть списком, производя несколько наклонов.

В общем случае, каждый элемент x имеет размерность {. $$"r y, где r есть ранг u, therefore specifying the increment in each possible direction. Аргумент x меньшего ранга, расширяется обычным способом. Например, x=: 1e_8 дает одинаковое приращение в каждом из направлений и дает приближение к производной.

```
   log=: ^.
   y=: 2 3 4
   1 log D:1 y
0.405465 0.287682 0.223144

   incr=: 1 0.1 0.01 1e_8
   incr log D:1/y
0.405465 0.287682 0.223144
0.487902 0.327898 0.246926
0.498754 0.332779 0.249688
     0.5 0.333333     0.25

   log D.1 y
0.5 0.333333 0.25
   %y
0.5 0.333333 0.25

   f=: +/@:*:"1
   g=: +/@:*:\"1
   (f y) ; (1 f D:1 y) ; (1 0.1 1e_8 f D:1 y)
+--+-----+--------------+
|29|5 7 9|  5  0.61 8e_8|
```

```
|  |     | 50   6.1 8e_7|
|  |     |5e8 6.1e7    8|
+--+-----+--------------+

   (g y) ; (1 g D:1 y)
+-------+-----+
|4 13 29|5 0 0|
|       |5 7 0|
|       |5 7 9|
+-------+-----+
```

Найти в Обломках `e.` — — — Содержит

`e.y` производит булевский результат, который определяет для каждого атома `y` -- содержит ли его распаковка элементы поломанного `y` .

Если `x` имеет размерность элемента `y` , то `x e. y` дает `1` , если `x` совпадает с некоторым элементом `y` . В общем случае, `x e. y` ↔ `(#y)>y i. x` .

Погрешность сравнения можно изменить настройкой, как в `e.!.t` .

Например:

```
   ]y=:'abc';'dc';'a'
+---+--+-+
|abc|dc|a|
+---+--+-+

   ;y
abcdca

   e. y
1 1 1 0 1 1
0 0 1 1 1 0
1 0 0 0 0 1

   f=: ] e.~&>/ ;
   f y
1 1 1 0 1 1
0 0 1 1 1 0
1 0 0 0 0 1

   'cat' e. 'abcd'
1 1 0

   ]z=: 2 3$'catdog'
cat
dog

   'cat' e. z
1
```

E. _ _ Найти

Единицы в x E. y обозначают начала последовательностей элементов x в y .

Например:

```
   'co' E. 'cocoa'
1 0 1 0 0

   s #~ -. '**' E. s=: 'Remove***multiple**stars.'
Remove*multiple*stars.

   ] x =: 0 1 2 ,: 2 3 4
0 1 2
2 3 4

   ] y=: 5 | i. 5 7
0 1 2 3 4 0 1
2 3 4 0 1 2 3
4 0 1 2 3 4 0
1 2 3 4 0 1 2
3 4 0 1 2 3 4

   x E. y
1 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 0 0

   ($x) x&-: ;. 3 y
1 0 0 0 0 0 0
0 0 0 1 0 0 0
0 1 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 0 0
```

Фиксируя `m f.` `u f.`

Если x выражение, то y=: `x f.` ему эквивалентно, но все встречающиеся в определении x имена (рекурсивно) заменяются их значениями. Последующее изменение значений этих имен, которое могло бы изменить определение x , не отражается на определении y .

Если x имя любого обьекта (существительного, глагола, наречия или союза), то `'x' f.` эквивалентно ему, но с рекурсивной заменой всех имен их значениями.

`x f.` не будет фиксировать любую часть x , содержащую `$:` .

Например:

```
   sum=: +/
   mean=: sum % #
   norm=: - mean
   norm a=: 2 3 4 5
_1.5 _0.5 0.5 1.5

   N=: norm f.
   N a
_1.5 _0.5 0.5 1.5

   norm
- mean

   N
- (+/ % #)

   sum=: -/
   norm a
2.5 3.5 4.5 5.5

   N a
_1.5 _0.5 0.5 1.5

   adv=: norm@
   *: adv
norm@*:
   adv
norm@

   'adv' f.
(- (-/ % #))@

   'a' f.
```

2 3 4 5

Гипергеометрическая ф-я `m H. n 0 0 0`

Союз `H.` применяется к двум числовым спискам и производит монаду, являющуюся гипергеометрической функцией, определенной в Главе 5 Abramowitz and Stegun [13]; она является пределом своего диадного случая, левый аргумент которого указывает требуемое количество членов в гипергеометрическом ряду.

Как обсуждается в Iverson [14], этот союз определен следующим образом:

```
   rf=: 1 : '(,m) ^!.1/ i.@['          NB. Нарастающий факториал
   L1=: 2 : 'm rf %&(*/) n rf'
   L2=: (i.@[ ^~ ]) % !@i.@[
   H =: 2 : '(m L1 n +/ . * L2) " 0'
```

Например:

```
   'a b'=: 2 3 5; 6 5

   a L1 b
(2 3 5 ^!.1/ i.@[) %&(*/) 6 5 ^!.1/ i.@[

   t=: 4 [ z=: 7

   t a L1 b z
1 1 1.71429 4.28571

   t (a H b , a H. b) z
295 295

   8 (1 H. 1) i. 6
1 2.71825 7.38095 19.8464 51.8063 128.619
   (1 H. 1) i. 6
1 2.71828 7.38906 20.0855 54.5982 148.413
   ^ i. 6
1 2.71828 7.38906 20.0855 54.5982 148.413

   erf   =: (1 H. 1.5)@*: * 2p_0.5&* % ^@:*:    NB. функция ошибок
   n01cdf=: -: @: >: @: erf @: ((%:0.5)&*)   NB. ИФР нормального распр. 0,1

   erf 0.5 1 1.5
0.5205 0.842701 0.966105
   n01cdf _2 _1.5 _1 _0.5 0 0.5 1 1.5 2
0.0227501 0.0668072 0.158655 0.308538 0.5 0.691462 0.841345 0.933193 0.97725
```

Целые i. 1 _ _ Индекс В

Размерность i.y равна |y , а содержимое представляет из себя первые */|y неотрицательных целых. Отрицательный элемент в y обозначает, обращение порядка элементов вдоль соответствующего измерения. Например:

```
   i. 5
0 1 2 3 4

   i. 2 _5
4 3 2 1 0
9 8 7 6 5
```

Если ранг элемента в x равен rix, тогда размерность результата x i. y есть (-rix)}.$y . Каждый атом результата есть либо #x, либо индекс первого вхождения в x соответствующей rix-ячейки y .

Сравнение в x i. y производится с погрешностью, которую можно *настроить*, как в i. !. t .

```
   (i.4);(i._4);(i.2 3 4);(i.2 _3 4);(i.'')
+-------+-------+-----------+-----------+-+
|0 1 2 3|3 2 1 0| 0  1  2  3| 8  9 10 11|0|
|       |       | 4  5  6  7| 4  5  6  7| |
|       |       | 8  9 10 11| 0  1  2  3| |
|       |       |           |           | |
|       |       |12 13 14 15|20 21 22 23| |
|       |       |16 17 18 19|16 17 18 19| |
|       |       |20 21 22 23|12 13 14 15| |
+-------+-------+-----------+-----------+-+

   A=: 'abcdefghijklmnopqrstuvwxyz'
   (A i. 'Now');(A i. 'now');(A {~ A i. 'now')
+--------+--------+---+
|26 14 22|13 14 22|now|
+--------+--------+---+

   m=: 5 4 $ 12{. A
   m;(m i. 'efgh');(1{m);(4{m)
+----+-+----+----+
|abcd|1|efgh|efgh|
|efgh| |    |    |
|ijkl| |    |    |
|abcd| |    |    |
|efgh| |    |    |
+----+-+----+----+
```

Лестница `i: 0 _ _` Индекс с Конца B

`i: y` есть список целых от `-y` до `y ;` в общем случае, `i: a j. b` производит список чисел от `-a` до `a` разделенных `b` равными промежутками (или `1+2*a` промежутками, если `b` равно `0`). Тоесть:

```
   i: 3
_3 _2 _1 0 1 2 3

   i: _2
2 1 0 _1 _2

   i: _2.5j4
2.5 1.25 0 _1.25 _2.5

   i: 0
0
```

`i:` работает как `i.` , только дает индекс *последнего* вхождения. Тоесть:

```
   1 2 3 4 1 2 3 i: 1 2 2 3 3 3 4 4 5
4 5 5 6 6 6 3 3 7

   1 2 3 4 1 2 3 i. 1 2 2 3 3 3 4 4 5
0 1 1 2 2 2 3 3 7
```

Например:

```
   (3 # i.3 4) i: (i.2 4)
2 5

   (3 # i.3 4) i. (i.2 4)
0 3

   () #~ i.@# = i.~) 'eleemosynary'     NB. первое вхождение каждой буквы
elmosynar

   () #~ i.@# = i:~) 'eleemosynary'     NB. последнее вхождение каждой буквы
lemosnary

   () #~ i.~ = i:~) 'eleemosynary'      NB. буквы, встречающиеся только один раз
lmosnar
```

`I. y` ↔ `(# i.@#) y`	x предполагается отсортированным по возрастанию или убыванию; если все элементы x одинаковы, предполагается, что они отсортированы по возрастанию. Сравнение производится без использования погрешности. Если y имеет размерность элемента x , то `x I. y` есть наименьшее неотрицательное `j` , такое что `j{x` следует за y , в соответствии с правилами упорядочения; или `#x` , если y следует за `{:x` или x не содержит элементов. Для y произвольного ранга, поиск применяется к ячейкам ранга `0>.(#$x)-1` .

Например:

```
   I. 0 0 1 0 1 0
2 4

   ] x=: ?. 10$20
6 15 19 12 14 19 0 17 0 14
   10 I.@:< x
1 2 3 4 5 7 9

   0 2 2 5 I. _5 1 2 3 9 0
0 1 1 3 4 0

   histogram=:
```

Мнимое j. 0 0 0 Комплексное

```
j. y  &harr;  0j1 * y
```

```
x j. y  &harr;  x + j. y
```

Например:

```
   j. 4
0j4

   3 j. 4
3j4

   a=: i. 3 3
   a;(j. 2*a);(a j. 2*a)
+-----+-------------+-------------+
|0 1 2|   0  0j2  0j4|   0  1j2  2j4|
|3 4 5| 0j6  0j8 0j10| 3j6  4j8 5j10|
|6 7 8|0j12 0j14 0j16|6j12 7j14 8j16|
+-----+-------------+-------------+

   (+ a j. 2*a);(|a j. 2*a)
+-----------------+----------------------+
|    0  1j_2  2j_4|      0 2.23607 4.47214|
| 3j_6  4j_8 5j_10| 6.7082 8.94427 11.1803|
|6j_12 7j_14 8j_16|13.4164 15.6525 17.8885|
+-----------------+----------------------+
```

```
   1 2 3 j./ 4 5 6 7
1j4 1j5 1j6 1j7
2j4 2j5 2j6 2j7
3j4 3j5 3j6 3j7

   j./?. 2 3 4$1000
146j713 755j318  79j151  852j92
854j178 439j260  660j90 257j862
 60j631 594j116 246j960 478j564
```

Таблица случайных комплексных чисел

Сосчитать Уровни

L. _

Если y не упаковочный или пуст, `L.` y равно `0`; иначе, результат равен единице плюс максимальная вложенность упакованных элементов.

Например:

```
   ]y=: (<<2 3 4),<(5 6 ; <<i. 2 3)
+-------+-------------+
|+-----+|+---+-------+|
||2 3 4|||5 6|+-----+||
|+-----+||   ||0 1 2|||
|       ||   ||3 4 5|||
|       ||   |+-----+||
|       |+---+-------+|
+-------+-------------+

   L. y
3

   L."0 y
2 3
```

На Уровне `u L: n _ _ _`

Союз `L:` применяет глагол `u` на уровнях, перечисленных `n` . Правый аргумент `n` ведет себя во многих отношениях как правый аргумент союза "С Рангом":

- Может содержать до трех элементов, указывающих уровни для монадного, левого и правого случаев.
- Полная форма указания уровней получается при помощи `3$&.|.n` . Например, если задан 2-х элементный список `p,q` , то это эквивалентно `q,p,q` .
- Отрицательные значения являются дополнительными: `u L:(-r) y` ↔ `u L:(0>.(L.y)-r) y`

Например:

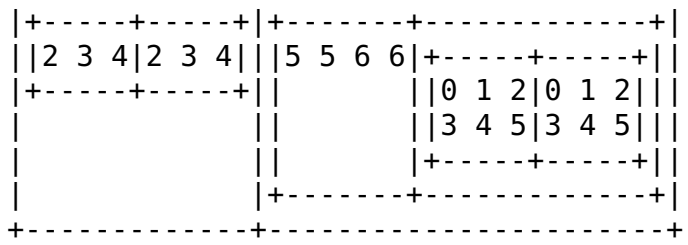
```
   ] y=: (<<2 3 4),<(5 6 ; <<i. 2 3)
+-------+-------------+
|+-----+|+---+-------+|
||2 3 4|||5 6|+-----+||
|+-----+||   ||0 1 2|||
|       ||   ||3 4 5|||
|       ||   |+-----+||
|       |+---+-------+|
+-------+-------------+
```

```
   +: L: 0 y
+-------+----------------+
|+-----+|+-----+--------+|
||4 6 8|||10 12|+------+||
|+-----+||      ||0 2  4|||
|       ||      ||6 8 10|||
|       ||      |+------+||
|       |+-----+--------+|
+-------+----------------+
```

Наречие `L:0` можно назвать *leaf* (к листьям)

```
   2 # L: 0 y
+-------------+-----------------+
|+-----------+|+-------+-------+|
||2 2 3 3 4 4|||5 5 6 6|+-----+||
|+-----------+||       ||0 1 2|||
|             ||       ||0 1 2|||
|             ||       ||3 4 5|||
|             ||       ||3 4 5|||
|             ||       |+-----+||
|             |+-------+-------+|
+-------------+-----------------+
```

```
   2 # L: 1 y
+-------------+-----------------------+
```


2 3 4 2 3 4
5 5 6 6
0 1 2 0 1 2
3 4 5 3 4 5

Запоминая u M. mu lu ru

Глагол u M. идентичен u , но способен хранить историю предыдущих аргументов и соответствующих им результатов для повторного использования. Обычно применяется в определениях многократно-рекурсивных глаголов.

Следующие примеры иллюстрируют пользу запоминания. fib n дает n-е число Фибоначчи. pn находит количество разбиений целого, используя рекурсивное соотношение Эйлера (уравнение 11 в http://mathworld.wolfram.com/PartitionFunctionP.html) .

```
fib=: 3 : 0 M.
 if. 1>:y do. y else. (fib y-1)+fib y-2 end.
)

fibx=: 3 : 0
 if. 1>:y do. y else. (fibx y-1)+fibx y-2 end.
)

   timer=: 6!:2

   timer 'fib 32'
0.000479377
   timer 'fibx 32'
43.696

   fib"0 i.18
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597

pn =: -/@(+/)@:($:"0)@rec ` (x:@(0&=)) @. (0>:]) M.
pnx=: -/@(+/)@:($:"0)@rec ` (x:@(0&=)) @. (0>:])
rec=: - (-: (*"1) _1 1 +/ 3 * ]) @ (>:@i.@>.@%:@((2%3)&*))

   timer 'pn 28'
0.000675355
   timer 'pnx 28'
61.7146

   pn"0 i.18
1 1 2 3 5 7 11 15 22 30 42 56 77 101 135 176 231 297
   pn 1000
24061467864032622473692149727991
```

Повторное применение запоминающего глагола к старым аргументам производит результат быстро:

```
   timer 'fib 32'
2.62393e_5
```

```
   timer 'pn 28'
4.01456e_5
```

M. применимо к анонимным глаголам и глаголам, производящим не-атомные результаты. Таким образом:

```
   timer '+/@:($:"0)@:(-&1 2)`]@.(1>:]) M. 32'
0.000186387
   timer '+/@:($:"0)@:(-&1 2)`]@.(1>:])    32'
8.61349

comb=: 4 : 0 M.   NB. Все сочетания x элементов из i.y
 if. (x>:y)+.0=x do. i.(x<:y),x else. (0,.x comb&.<: y),1+x comb y-1 end.
)

   3 comb 5
0 1 2
0 1 3
0 1 4
0 2 3
0 2 4
0 3 4
1 2 3
1 2 4
1 3 4
2 3 4
```

Комментарий NB.

Остаток строки после NB. игнорируется.

Например:

```
   text=: 'i. 3 4 NB. 3-by-4 table'
   ;: text
+--+---+----------------+
|i.|3 4|NB. 3-by-4 table|
+--+---+----------------+

   ". text                      NB. Выполнить текст
0 1  2  3
4 5  6  7
8 9 10 12
```

Умножить на Пи

`o.  0 0 0`

Круговая функция

`o. y` возвращает π умноженное на `y` . Тоесть `o. 1` равно приблизительно `3.14159` .

Функция `x&o.` является четной или нечетной в зависимости от четности `x`. Функция `(-x)&o.` обратна к `x&o.` (тоесть, `y = (-x) o. x o. y` для существенного подмножества значений `y`).

x o. y	x	(-x) o. y	
Sqrt 1-(Sqr y)	**0**	Sqrt 1-(Sqr y)	`0&o.@(1&o.)` ↔ `2&o.`
Sine y	**1**	Arcsine y	аргумент/результат в радианах
Cosine y	**2**	Arccos y	"
Tangent y	**3**	Arctan y	"
Sqrt (Sqr y)+1	**4**	Sqrt (Sqr y)-1	`4&o.@(5&o.)` ↔ `6&o.`
Sinh y	**5**	Arcsinh y	Sinh -- гиперболический синус
Cosh y	**6**	Arccosh y	
Tanh y	**7**	Arctanh y	
Sqrt - (1 + Sqr y)	**8**	- Sqrt - (1 + Sqr y)	
RealPart y	**9**	y	
Magnitude y	**10**	Conjugate (+y)	
ImaginaryPart y	**11**	`j. y`	
AngleOf y	**12**	`^j. y`	

Примеры:

```
   rfd=: 180 %~ o.                NB. Перевести градусы в радианы
   sin=: 1&o.
   SIN=: sin@rfd
   (rfd 0 90 180);(sin 0 1.5708);(SIN 0 90)
+---------------+---+---+
```

```
|0 1.5708 3.14159|0 1|0 1|
+----------------+---+---+
```

```
  p. c   &harr; (m;r)
  p.p. c &harr;  c
```

Если e вектор неотрицательных целых, то p.<c,.e дает коэффициенты соответствующего многочлена:

```
  (p. <c,.e)&p. &harr; (<c,.e)&p.
```

Есть три представления: коэффициенты (c); множитель (m) и корни (r); упакованная матрица коэффициентов (c) и соответствующих степеней (e), т.е. мультиномиальный многочлен. Для них:

```
      c p. x  &harr;  +/c*x^i.#c
  (m;r) p. x  &harr;  m * */x-r
     (<r)&p.  &harr;  (1;r)&p.
 (<c,.e)p.<y  &harr;  c+/ .*e*/ .(^~)y
```

где m -- скаляр; c и r -- скаляры или векторы; e -- вектор или матрица, такая что ($e)-:(#c),(#y) . Для скалярного y доопределение производится обычным образом.

```
   p. 1 0 0 1
+-+----------------------------+
|1|_1 0.5j0.866025 0.5j_0.866025|
+-+----------------------------+

   ]mr=: p. c=: 0 16 _12 2     NB. Множитель/Корни из Коэффициентов
+-+-----+
|2|4 2 0|
+-+-----+

   x=: 0 1 2 3 4 5
   (c p. x), ((<c,.i.4)p. x), (mr p. x),: 2*(x-4)*(x-2)*(x-0)
0 6 0 _6 0 30
0 6 0 _6 0 30
0 6 0 _6 0 30
0 6 0 _6 0 30

   c=: 1 3 3 1
   c p. x
1 8 27 64 125 216
   (x+1)^3
1 8 27 64 125 216
```

```
   bc=: !~/~i.5                NB. Биномиальные коэффициенты
   bc;(bc p./ x);((i.5) ^~/ x+1)
+---------+--------------------+--------------------+
|1 0 0 0 0|1  1  1   1   1    1|1  1  1   1   1    1|
|1 1 0 0 0|1  2  3   4   5    6|1  2  3   4   5    6|
|1 2 1 0 0|1  4  9  16  25   36|1  4  9  16  25   36|
|1 3 3 1 0|1  8 27  64 125  216|1  8 27  64 125  216|
|1 4 6 4 1|1 16 81 256 625 1296|1 16 81 256 625 1296|
+---------+--------------------+--------------------+

   c&p. d. 1 x                 NB. Первая производная многочлена
3 12 27 48 75 108

   (<1 _1 ,. 5 0) p. 3         NB. Коэффициенты/Степени
242

   _1 0 0 0 0 1 p. 3
242

   p. <1 _1 ,. 5 0             NB. Коэффициенты/Степени в Коэффициенты
_1 0 0 0 0 1

   c=: _1 1 2 3 [ e=: 4 2$2 1 1 1 1 2 0 2
   c,.e                        NB. Коэффициенты/Степени
_1 2 1
 1 1 1
 2 1 2
 3 0 2

   (<c,.e) p. <y=:2.5 _1       NB. Мультиномиальный многочлен
11.75

   c +/ .* e */ .(^~) y
11.75
```

Заметьте, что `(<c,.e)p.<y` является "правильным" мультиномиальным многочленом только если элементы `e` являются неотрицательными целыми. В общем случае, степени так не ограничены, как, например, во взвешенной сумме квадратного корня и корня четвертой степени:

```
   ] t=: <2 3,.1r2 1r4
+-----+
|2 1r2|
|3 1r4|
+-----+

   (t p. 16), +/ 2 3 * 16 ^ 1r2 1r4
14 14
```

Вариант `p.!.s` представляет из себя *лестничный* многочлен; он отличается от `p.` тем, что его определение основывается на лестнице `^!.s` , вместо `^` (степени).

Производная M. `p.. 1 0 1` Интеграл M.

В применении к многочлену (коэффициенты или упакованные корни), `p..` дает коэффициенты производной многочлена. Например:

```
   p.. 1 2 3 4 5
2 6 12 20
   p.. 5 4 3 2 1
4 6 6 4

   p.. 2; 1j1 1j_1
_4 4
   p.. p. 2; 1j1 1j_1
_4 4
```

`x p.. y` дает коэффициенты итеграла от многочлена `y` с постоянной интегрирования, равной `x` . Тоесть:

```
   5 p.. 4 6 6 4
5 4 3 2 1
   1 p.. 2 6 12 20
1 2 3 4 5
```

Дальнейшие примеры:

```
   p.. 1&o. t. i. 11x              NB. производная синуса
1 0 _1r2 0 1r24 0 _1r720 0 1r40320 0
   2&o. t. i.10x                   NB. косинус
1 0 _1r2 0 1r24 0 _1r720 0 1r40320 0

   p.. 2&o. t. i. 11x              NB. производная косинуса
0 _1 0 1r6 0 _1r120 0 1r5040 0 _1r362880
   -@(1&o.) t. i.10x               NB. минус синус
0 _1 0 1r6 0 _1r120 0 1r5040 0 _1r362880
   - (1&o. t. i.10x)
0 _1 0 1r6 0 _1r120 0 1r5040 0 _1r362880

   p..^:(i.@#) 8 $ 1
   1    1    1   1   1  1 1 1
   1    2    3   4   5  6 7 0
   2    6   12  20  30 42 0 0
   6   24   60 120 210  0 0 0
  24  120  360 840   0  0 0 0
 120  720 2520   0   0  0 0 0
 720 5040    0   0   0  0 0 0
5040    0    0   0   0  0 0 0
```

Результатом p: i является i-тое простое число. Например:

```
   p: 0
2
   p: i. 15
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

Обратная функция к p: есть количество простых чисел, меньших некоторого заданного числа (часто обозначается как π(n)):

```
   pi=: p:^:_1
   pi i. 15
0 0 0 1 2 2 3 3 4 4 4 4 5 5 6

   (] , pi ,: p:@pi) i.15
0 1 2 3 4 5 6 7  8  9 10 11 12 13 14
0 0 0 1 2 2 3 3  4  4  4  4  5  5  6
2 2 2 3 5 5 7 7 11 11 11 11 13 13 17

   y=: (2^31)-1
   ]a=: pi y
105097564
   ]b=: p: a
2147483647
   b=y
1
```

4 p: y простое число, предшествующее y ; эквивалентно <:^:(0&p:)^:"0 >.y-1 .

_1 p: y эквивалентно p:^:_1 y : количество простых, меньших y .

0 p: y эквивалентно -.1 p: y .

1 p: y равно 1 тогда и только тогда, когда y простое.

2 p: y эквивалентно __ q: y , дает 2-х строчную таблицу множителей и их степеней в разложении y на простые множители.

3 p: y эквивалентно q: y , список простых множителей y , произведение которых равно y .

4 p: y -- простое число, следующее за y ; эквивалентно >:^:(0&p:)^:_"0 <.1+y .

5 p: y вычисляет фи-функцию Эйлера (или "тотиенту", Euler’s totient function) от аргумента y : количество неотрицательных целых меньших y и, при этом, относительно к нему простых, тоесть сумму +/1=y+.i.y .

На данный момент, аргументы, большие 2^31, проверяются на простоту стохастическим алгоритмом Миллера-Рабина (Miller-Rabin).

```
   4 p: 20
23
   4&p:^:(i.8) 20
20 23 29 31 37 41 43 47

   _4 p: 50
47
   _4&p:^:(i.8) 50
50 47 43 41 37 31 29 23

   ] y=: !14x
87178291200
   ] c=: _4 4 p:"0 y
87178291199 87178291219
   1 p: ({.c)+i.1--/c
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
   1 p: _1+2^67x
0
   m=: 366384x * */ p: i.9x
   1 p: 6171054912832631x + m * i.25
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

   2 p: y
 2 3 5 7 11 13
11 5 2 2  1  1
   'p e'=: 2 p: y
   */p^e
87178291200
   y
87178291200

   3 p: y
2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 5 5 7 7 11 13
   */ 3 p: y
87178291200
   y
87178291200
   e#p
2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 5 5 7 7 11 13

   5 p: y
16721510400
   */(p-1)*p^e-1
16721510400
   (- ~:)&.q: y
16721510400
```

Простые Множители q: 0 0 0 Степени Простых

q: y есть список простых множителей в разложении положительного целого аргумента y . Например:

```
   y=: 105600
   q: y
2 2 2 2 2 2 3 5 5 11
   */ q: y
105600
   $ q: 1
0
   */ q: 1
1
   q: b. _1
*/
   q: _1+2^67x
193707721 761838257287
```

Если x положительно и конечно, x q: y есть список степеней простых чисел в разложении положительного целого y на простые множители, соответствующие первым x простым числам; при x равном _ , берется достаточное для представления аргумента количество простых чисел.

Для отрицательных и конечных x , x q: y есть таблица из двух строк, содержащая последние |x простых чисел и их степени в разложении y на простые множители ; при x равном __ , берется достаточное для представления аргумента количество простых. Например:

```
   2 q: 700
2 0
   10 q: 700
2 0 2 1 0 0 0 0 0 0
   _ q: 700
2 0 2 1
```

В текущей реализации, множители в результате q: , большие 2^31 , проверяются на простоту стохастическим алгоритмом Миллера-Рабина (Miller-Rabin).

```
   ~.@q: 700                          NB. Различные простые множители
2 5 7

   +/"1@=@q: 700                      NB. Степени в разложении
2 2 1

   5 q: 144^100x
400 200 0 0 0

   __ q: 700
```

```
2 5 7
2 2 1

   __ q: !20x
 2 3 5 7 11 13 17 19
18 8 4 2  1  1  1  1

   y=: 100
   e=: _&q:                                NB. Полный список степеней
   (e ; +:&.e ; -:&.e ; %&3&.e)y           NB. Степени, квадрат, корень, кубический
+-----+-----+--+-------+
|2 0 2|10000|10|4.64159|
+-----+-----+--+-------+

   V=: /@,:                                NB. Для векторов несовместимой длины

   12 (+V&.e; -V&.e; >.V&.e; <.V&.e) y     NB. Произведение, остаток, НОК, НОД
+----+----+---+-+
|1200|0.12|300|4|
+----+----+---+-+

   totient=: * -.@%@~.&.q:                 NB. Функция Эйлера
   totient 700
240

   +/ 1 = 700 +. i.700
240
```

Направление r. 0 0 0 Полярные

r.y ↔ ^j.y	x r. y ↔ x*r. y

Результат r. y есть комплексное число с модулем 1 , мнимая и действительная части которого соответствуют координатам точки на единичном круге, отклоненной от оси абсцисс на y радиан. Например:

```
   r. 2
_0.416147j0.909297

   +. r. 2
_0.416147 0.909297

   | r. 2
1

   y=: 1r4 * o. i.7            NB. Кратные четверти &pi;
   format=: 8j3&":
   (format ,.y);(format +. r.y);(format +. 2 r.y)
+--------+----------------+----------------+
|   0.000|   1.000   0.000|   2.000   0.000|
|   0.785|   0.707   0.707|   1.414   1.414|
|   1.571|   0.000   1.000|   0.000   2.000|
|   2.356|  _0.707   0.707|  _1.414   1.414|
|   3.142|  _1.000   0.000|  _2.000   0.000|
|   3.927|  _0.707  _0.707|  _1.414  _1.414|
|   4.712|   0.000  _1.000|   0.000  _2.000|
+--------+----------------+----------------+

   3 r. _2
_1.24844j_2.72789

   *. 3 r. _2
3 _2
```

Символ `s:` _ _ _ Символ

Символы -- тип данных, производимый глаголом `s:` . Поиск, сортировка и сравнение символов гораздо быстрее соответствующих операций с обычными упакованными строками. Структурные, поисковые и устанавливающие отношение глаголы работают и с символами. Арифметические глаголы с символами не работают.

Монада `s:` производит массив символов. Допустимы несколько типов аргументов:

- строка, в которой первый символ служит разделителем
- текстовый массив, в котором каждая строка, исключая пробелы в конце, представляет собой имя символа
- упаковочный массив строк

`s:^:_1`, обратный к `s:`, есть `5&s:` .

Диада `s:` принимает скалярный целый левый аргумент и вычисляет множество различных функций:

Левый	Правый	Функция
`0`	`0`	мощность (cardinality) множества символов
`0`	`1`	длина строки (количество букв, занятых в массиве строк)
`0`	`2`	таблица символов; столбцы соответствуют: 0 индекс в таблице строк 1 длина в байтах 2 значение хеш-функции 3 цвет 4 родитель 5 левый 6 правый 7 порядковый # 8 предыдущий 9 следующий 10 флажки

		Детали этих данных могут меняться от одной версии J к другой.
0	3	таблица строк
0	4	хеш-таблица. _1 соответствует свободным элементам; неотрицательные значения -- индексы в таблицу символов.
0	5	корень дерева двоичного поиска
0	6	коэффициент заполнения дерева двоичного поиска
0	7	промежуток дерева двоичного поиска
0	10	установить глобальные данные о символах, эквивалентно `0 s:&.>i.8`. (детали этих данных могут измениться в будущем).
0	11	провести контроль целостности глобальных данных о символах
0	12	количество запросов для нахождения каждого символа
1	массив символов	строка имен символов, предшествуя каждое знаком
_1	строка	список символов для строки, содержащей имена символов, каждое, с префиксом из первого символа строки
2	массив символов	строка имен символов, каждое имя заканчивается нулем (null-terminated)
_2	строка	список символов для строки, содержащей имена символов, каждое из которых имеет суффикс из последнего символа строки
3	массив символов	текстовый массив имен символов, выровненных

		нулем (ASCII код = 0)
_3	текстовый массив	массив символов для текстового массива, где каждая строка есть имя символа, исключая нули (ASCII код = 0) в конце
4	массив символов	текстовый массив имен символов, выровненных пробелами
_4	текстовый массив	массив символов для текстового массива, каждая строка которого содержит (исключая пробелы в конце) имя символа
5	массив символов	массив имен символов в виде массива упакованных строк
_5	упакованые строки	массив символов для массива упакованных строк, каждая из которых представляет собой имя символа
6	массив символов	массив целых индексов символов (индексы в таблицу символов)
_6	индексы	символы по индексам
7	массив символов	масив целых порядковых номеров символов
10	глобальные данные символов	установить глобальные данные символов (в форме, возвращенной `0 s: 10`) после проверки ее целостности. Испорченные данные могут вызвать неправильную интерпретацию массивов символов, потерю данных, сбой в системе, или конец известной нам цивилизации.

Обратный к `k&s:` глагол есть `(-k)&s:`, для ненулевых целых `k` между `_6` и `6` .

Остаток этого текста разбит на следующие разделы: Отображение, Комментированные примеры, Память и процессорное время, and Длительное хранение.

Отображение

Символы отображаются в виде `` (96{a.) , присоединенного к началу имени символа; отображение массивов символов подобно отображению числовых массивов, за исключением того, что столбцы выравниваются *влево*. См. Комментированные примеры ниже.

Комментированные примеры

```
   ] t=: s: ' zero one two three four five'
`zero `one `two `three `four `five

   $ t                              NB. список 6-ти символов
6
   3 5 $ t                          NB. матрица символов
`zero `one  `two  `three `four
`five `zero `one  `two   `three
`four `five `zero `one   `two

   1 3 5 3 1 { t
`one `three `five `three `one
   |. t
`five `four `three `two `one `zero
   _2 |. t
`four `five `zero `one `two `three
   1 0 2 0 4 0 # t
`zero `two `two `four `four `four `four

   <"0 t                            NB. символы могут быть упакованы
+-----+----+----+------+-----+-----+
|`zero|`one|`two|`three|`four|`five|
+-----+----+----+------+-----+-----+
   (2|i.#t) </. t
+----------------+-----------------+
|`zero `two `four|`one `three `five|
+----------------+-----------------+

   <:/~ t                           NB. отношения работают для символов
1 0 0 0 0 0
1 1 1 1 0 0
1 0 1 0 0 0
1 0 1 1 0 0
1 1 1 1 1 0
1 1 1 1 1 1

   t + t                            NB. арифметические функции не работают
|domain error
|   t    +t

   /: t                             NB. символы могут быть упорядочены
```

```
5 4 1 3 2 0

   5 s: t                           NB. преобр. символы в упакованные строки
+----+---+---+-----+----+----+
|zero|one|two|three|four|five|
+----+---+---+-----+----+----+
   (/: t) -: /: 5 s: t
1

   /:~ t
`five `four `one `three `two `zero

   <:/~ /:~ t
1 1 1 1 1 1
0 1 1 1 1 1
0 0 1 1 1 1
0 0 0 1 1 1
0 0 0 0 1 1
0 0 0 0 0 1

   t i.  s: ' three one four one five nine'
3 1 4 1 5 6
   t e.~ s: ' three one four one five nine'
1 1 1 1 1 0

   10{. t                           NB. заполнитель есть символ нулевой длины
`zero `one `two `three `four `five ` ` ` `

   _10{.t
` ` ` ` `zero `one `two `three `four `five

   0 s: 0                           NB. мощность (текущее # уникальных символов)
8

   a=:   ;:'A AAPL AMAT AMD AMZN ATT BA CRA CSCO DELL F GE GM HWP IBM INTC'
   a=: a,;:'JDSU LLY LU MOT MSFT NOK NT PFE PG QCOM RMBS T XRX YHOO'
   b=: ;:'NY SF LDN TOK HK FF TOR'
   c=: ;:'Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec'
   d=: <;._1 ' 00 01 02 03 04 05 06 07 08 09'
   e=: ;:'open high low close'

   t=: }.@;&.>{' ',&.>&.>a;b;c;d;<e
   $t
30 7 12 10 4
   */ $t
100800
   2 4 ($,) t
+----------------+----------------+---------------+-----------------+
|A NY Jan 00 open|A NY Jan 00 high|A NY Jan 00 low|A NY Jan 00 close|
+----------------+----------------+---------------+-----------------+
|A NY Jan 01 open|A NY Jan 01 high|A NY Jan 01 low|A NY Jan 01 close|
+----------------+----------------+---------------+-----------------+
   y=: s: t                         NB. создать очень много символов
   $y
30 7 12 10 4
   2 4 ($,) y
`A NY Jan 00 open `A NY Jan 00 high `A NY Jan 00 low `A NY Jan 00 close
```

```
`A NY Jan 01 open `A NY Jan 01 high `A NY Jan 01 low `A NY Jan 01 close

   0 s: 11                          NB. проверить целостность таблиц
1
   0 s: 0                           NB. мощность
100808

   (+/ % #) 0 s: 12                 NB. среднее # запросов на символ
1.31213

   h=: 100808 {. 2 {"1 ] 0 s: 2     NB. хеш-значения

   (+/ ~: h) % #h                   NB. доля разных хеш-значений
0.999821
   (+/ ~: h |~ #0 s: 4) % #h        NB. доля по отношению к хеш-таблице
0.831005
```

Память и процессорное время

В текущей реализации символ y требует 4 байта для индекса 8 или более байт в хеш-таблице, 44 байта в таблице символов, и `len y` байт (умноженное на два, если Unicode) в таблице строк, где `len=: #&>@(5&s:)` есть длина имени символа. (символ требует один 4-х байтный элемент хеш-таблицы, но для эффективного хеширования система поддерживает по меньшей мере 2*n элементов для n символов.) Множественные вхождения символов требуют лишь дополнительных индексов; элементы хеш-таблицы, таблицы символов, и таблицы строк не дублируются.

Вычисления с символами обычно требуют линейного времени. В частности:

query (new)	`O((len y) * ^. 0 s: 0)`
query (old)	`O(len y)`
`/:y`	`O(*/$y)`
`i{y`	`O((*/$i) * */}.$y)`
`x < y` etc.	`O(x >.&(*/@$) y)`
`x i. y`	`O(x + &(*/@$) y)`

Длительное хранение

Интерпретация символов зависит от глобальных данных о символах `0 s: 10`. Чтобы сохранить эту интерпретацию для разных сессий J необходимо сохранить и восстановить их в начале сессии. Тоесть:

```
((3!:1) 0 s: 10) 1!:2 <'symb.dat'      сохранить символы
10 s: (3!:2) 1!:1 <'symb.dat'          восстановить символы
```

См. предостережения в описании `10 s: x`.

Извлекая Уровень

u S: n _ _ _

u S: n производит список, получающийся применением u к аргументу(-ам) на уровнях n (интерпретируется так же как правый аргумент L:). Например, #0:S:0 y есть количество листьев (массивов на уровне 0) in y .

Например:

```
   fetch=: >@({&>/)@(<"0@|.@[ , <@]) " 1 _
   ] t=: 5!:2 <'fetch'             NB. Массив интересной структуры
+-----------------------------------------------------+-+---+
|+-----------------+-+-------------------------------+|"|1 _|
||+-+-+-----------+|@|+---------------------+-+-------+|| |   |
|||>|@|+-------+-+|| ||+---------------+-+-+|,|+-+-+-+||| |   |
||| | ||+-+-+-+|/||| |||+-------+-+--+|@|[|| ||<|@|]||||| |   |
||| | |||{|&|>|| ||| ||||+-+-+-+|@||.|| | || |+-+-+-+||| |   |
||| | ||+-+-+-+| ||| |||||<|"|0|| |  || | || |       ||| |   |
||| | |+-------+-+|| ||||+-+-+-+| |  || | || |       ||| |   |
||+-+-+-----------+| |||+-------+-+--+| | || |       ||| |   |
||                 | ||+---------------+-+-+| |       ||| |   |
||                 | |+---------------------+-+-------+|| |   |
|+-----------------+-+-------------------------------+| |   |
+-----------------------------------------------------+-+---+

   < S: 0 t                        NB. упакованные листья t
+-+-+-+-+-+-+-+-+-+-+-+--+-+-+-+-+-+-+-+---+
|>|@|{|&|>|/|@|<|"|0|@||.|@|[|,|<|@|]|"|1 _|
+-+-+-+-+-+-+-+-+-+-+-+--+-+-+-+-+-+-+-+---+

   11 {. t (; <@;) S: 0 1 {:: t    NB. Таблица листьев и путей к ним
+--+-----------+
|> |0 0 0      |
+--+-----------+
|@ |0 0 1      |
+--+-----------+
|{ |0 0 2 0 0  |
+--+-----------+
|& |0 0 2 0 1  |
+--+-----------+
|> |0 0 2 0 2  |
+--+-----------+
|/ |0 0 2 1    |
+--+-----------+
|@ |0 1        |
+--+-----------+
|< |0 2 0 0 0 0|
+--+-----------+
|" |0 2 0 0 0 1|
+--+-----------+
```

```
|0 |0 2 0 0 0 2|
+--+-----------+
|@ |0 2 0 0 1  |
+--+-----------+
||.|0 2 0 0 2  |
+--+-----------+
```

Коэффициент Тейлора

`u t. 0 0 0`

`u t. y` есть y-й коэффициент в разложении Тейлора функции `u`. Область определения левого аргумента наречия `t.` та-же, что и левого аргумента производной `D.`. См. случай `m t.`.

`x u t.y` есть произведение `(x^y)` и `u t. y`.

Например:

```
   f=: 1 2 1&p.
   g=: 1 3 3 1&p.
   x=: 10%~i=: i.8
   ]c=: (f*g) t. i
1 5 10 10 5 1 0 0

   6j2 ":(c p. x),:(f*g) x
  1.00  1.61  2.49  3.71  5.38  7.59 10.49 14.20
  1.00  1.61  2.49  3.71  5.38  7.59 10.49 14.20

   (c p. x)=(f*g) x
1 1 1 1 1 1 1 1

   ]d=: f@g t. i
4 12 21 22 15 6 1 0

   (d p. x)=(f g x)
1 1 1 1 1 1 1 1

   sin=: 1&o.
   cos=: 2&o.
   8j4":t=: (^ t. i),(sin t. i),:(cos t. i)
  1.0000  1.0000  0.5000  0.1667  0.0417  0.0083  0.0014  0.0002
  0.0000  1.0000  0.0000 _0.1667  0.0000  0.0083  0.0000 _0.0002
  1.0000  0.0000 _0.5000  0.0000  0.0417  0.0000 _0.0014  0.0000

   * t
1 1  1  1 1 1  1  1
0 1  0 _1 0 1  0 _1
1 0 _1  0 1 0 _1  0

   ((sin*sin)+(cos*cos)) t. i
1 0 0 0 _2.71051e_20 0 0 0

   rf=: n%d
   n=: 0 1&p.
   d=: 1 _1 _1&p.
   ]fibonacci=: rf t. i. 20
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181

   2 +/\ fibonacci
1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

   (% -. - *:) t. i.20
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

Присвоить Ряд Тейлора

m t. 0 0 0

Функция h=: u`v t. эквивалентна u , за исключением того, что функция, производимая h t. есть v . Например, явное определение (такое как определение exp ниже) не раскладывается автоматически в ряд Тейлора, при помощи этого наречия разложение ему можно присвоить явно.

```
   y=: i. 5
   ^y
1 2.71828 7.38906 20.0855 54.5982

   ^ t. y
1 1 0.5 0.1666667 0.04166667

   exp=: 3 : '^ y'
   exp y
1 2.71828 7.38906 20.0855 54.5982

   exp t. y
|domain error
|       exp t.y

   e=: exp`(%@!) t.
   e y
1 2.71828 7.38906 20.0855 54.5982

   e t. y
1 1 0.5 0.1666667 0.04166667

   %@e t. y
1 _1 0.5 _0.1666667 0.04166667

   %@^ t. y
1 _1 0.5 _0.1666667 0.04166667
```

Коэффициент Тейлора (взвеш.)

```
u t:  0 0 0
```

Результатом `u t: k` является `(!k)*u t. k` . Другими словами, `t:` производит коэффициент Тейлора, *взвешенный* факториалом. Как следствие, в применении к функциям типа экспоненты, такие коэффициенты показывают простые закономерности. Разложения этого типа соответствуют *экспоненциальной производящей функции*.

Например:

```
   k=: i. 12
   ^ t: k
1 1 1 1 1 1 1 1 1 1 1 1

   %@^t: k                    NB. Спадающая экспонента
1 _1 1 _1 1 _1 1 _1 1 _1 1 _1

   sin =: 1&o.
   cos =: 2&o.
   sinh=: 5&o.
   cosh=: 6&o.
   exp=: ^
   dec=: %@^

   (exp t:,dec t:,sinh t:,cosh t:,sin t:,:cos t:) k
1  1  1  1 1  1  1  1 1  1  1  1
1 _1  1 _1 1 _1  1 _1 1 _1  1 _1
0  1  0  1 0  1  0  1 0  1  0  1
1  0  1  0 1  0  1  0 1  0  1  0
0  1  0 _1 0  1  0 _1 0  1  0 _1
1  0 _1  0 1  0 _1  0 1  0 _1  0
```

Приближение Тейлора u T. n

`u T. n` есть n-членное приближение Тейлора к функции `u` .

Например:

```
   6j2 ": ^ T. 8 x=: 2 %~ i.8
  1.00  1.65  2.72  4.48  7.38 12.13 19.85 32.23

   6j2 ": ^ x
  1.00  1.65  2.72  4.48  7.39 12.18 20.09 33.12

   ^ T. _
3 : 0"0
g=:p.&y@:((^) t.)@i.
g +:^:(g ~: g@+:)^:_ ] 1
)
   (^ = ^T._) i. 5
1 1 1 1 1
```

Сравните *союз* `T.` с *наречием* `t.` .

Уникод (Unicode) u: — — — Уникод (Unicode)

типы данных J:	*char* (1-байтный символ) — 8-битное значение от 0 до 255
	wchar (2-байтный символ, wide char) — 16-битное значение от 0 до 65535
Кодировки:	ASCII — 0 до 127, подмножество U8
	U8 — многобайтная кодировка символов Unicode

Большинство диад u: работают со значениями, не с кодировками. Кодировки ASCII и U8 используются в 7&u: и 8&u: .

Монада u: принимает несколько видов аргумента:

Аргумент	Результат
char	как 2&u:
wchar	копия аргумента
целые	как 4&u:

Обратным к u: является глагол 3&u:

Диада u: принимает скалярный целый левый аргумент и несколько видов правого аргумента:

Левый	Результат	Правый	
1	char	char	как есть
		wchar	отбрасыв старшие
2	wchar	char	старшие равны 0
		wchar	как есть
3	integers	char или wchar	
4	wchar	целые от -65536 65535	
5	char	wchar от 0 до 25	
6	wchar	пары char преобразуются в wchar	
7	char или wchar	U8	преобраз в wchar
		ASCII	как есть
		wchar	если все значения <128, преобраз в ASCII, и

			как есть пустой правый аргумент произв пустой массив ch
8	U8	wchar	преобраз в U8
		char	как есть пустой правый аргумент произв пустой массив ch

1&u: и 2&u: обратны друг другу, так же 3&u: и 4&u: , так же 7&u: и 8&u: .

Например:

```
   ] t=: u: 'We the people'
We the people
   3!:0 t
131072                          NB. код типа данных unicode есть 131072

   u: 97 98 99 +/ 0 256 512 1024
aaaa                            NB. 2-байтные символы показываются
bbbb                            NB. как 1-байтные
cccc

   'a' = u: 97 + 0 256 512 1024
1 0 0 0

   ] t=: (2 4$'abcdefgh') , u: 'wxyz'
abcd                            NB. списки 1- и 2-байтных символов можно соединять
efgh                            NB. 1-байтные символы при этом преобразуются.
wxyz
   3!:0 t
131072
```

Неограниченная Точность `x:  _ _ _` Числ./Знам.

`x:` в применении к действительным числам, производит рациональные дроби неограниченной точности. В применении к целым, производит целые неограниченной точности. Подразумеваемое `x:` сравнение производится с погрешностью. Обратный глагол `x:^:_1` преобразует рациональные дроби (в т.ч. и неограниченной точности) в числа ограниченной точности (действительные или целые).

`1 x: y` то-же, что и `x: y ; a 2 x: y` производит два целых неограниченной точности, соответствующих числителю и знаменателю аргумента.

`(-k)x: y` обращает `k x: y ;` тоесть, `_1 x: y` совпадает с `x:^:_1 y`, а `_2 x: y` с `%/"1 y`.

```
   x: 1.2
6r5

   2 x: 1.2
6 5

   x: 1.2 _1.2 0 0.07
6r5 _6r5 0 7r100

   x: 3j4
|domain error
|       x:3j4

   ] pi =: o.1
3.14159

   x: pi
1285290289249r409120605684

   pi - 1285290289249%409120605684
1.49214e_13

   x:!.0 pi
884279719003555r281474976710656

   pi - 884279719003555%281474976710656
0
```

```
   2 x: 1r2 3r4 5r6 _7r8
 1 2
 3 4
 5 6
_7 8
```

См. так же Раздел II G.

Постоянные Функции

от _9: до 9: _ _ _

Результат этих глаголов равен _9 , _8 , _7 и так далее до 9 .

Например:

```
   x=: 1 2 3 [ y=: 4 5 6
   2: y
2

   x 9: y
9
```

Союз *с рангом* " , в применении к любому существительному, производит из него постоянную функцию указанного ранга. Таким образом, вышеупомянутые постоянные функции можно записать и в виде i"_ . Например:

```
   2"_ y
2

   2"0 y
2 2 2

   2"1 i. 2 3 4
2 2 2
2 2 2

   2p1"0 y
6.28319 6.28319 6.28319

   1p1 1p_1 1x1"0 y          NB. Постоянная функция с векторным результатом
3.14159 0.3183099 2.71828
3.14159 0.3183099 2.71828
3.14159 0.3183099 2.71828

   0 0 0"1 y                 NB. Нулевой вектор в трехмерном пространстве
0 0 0

   a=:'abcdefghijklmnopqrstuvwxyz'"_
   A=:'ABCDEFGHIJKLMNOPQRSTUVWXYZ'"_
   s=: ' '"_

   f=: { a
   f 5 8 6
fig

   g=: { a,:A
   g 1 5;0 8;0 6
```

Fig

Константы

При записи числовых констант, иллюстрированной в Главе I, можно использовать и другие символы. Например: 2r3 означает две третьих, 2p1 два π, а 2e3p1 есть 2000 π. Интерпретация этих символов подчиняется следующей иерархии:

.	Десятичная точка имеет высший приоритет
_	Далее интепретируется знак "минус"
e	Потом десятичная степень (экспоненциальная запись)
ad ar j	Комплексные: модуль и аргумент в градусах (**ad**) или радианах (**ar**); действ. и мнимая части
p x	Величина по основанию Пи (**p**, равного o.1) и числа Эйлера (**x**, т.е. экспоненты ^1)
b	Числа в N-ричной системе счисления (буквы a - z обозначают цифры 10 - 35)

Кроме того, последовательность цифр, заканчивающаяся символом x , обозначает целое число неограниченной точности; а цифры до и после r интерпретируются как числитель и знаменатель рационального числа. См. Раздел II G.

Например, 2.3 обозначает две целых три десятых; _2.3 обозначает то-же с обратным знаком; но _2j3 обозначает комплексное число с действительной частью _2 и мнимой частью 3, а *не* комплексное число 2j3 с обратным знаком. Символы на одном уровне иерархии не могут быть использованы вместе (1p2x3 является ошибкой).

Следующие строки иллюстрируют основные моменты:

```
   2.3e2 2.3e_2 2j3
230 0.023 2j3

   2p1 1p_1
6.28319 0.31831

   1x2 2x1 1x_1
7.38906 5.43656 0.367879

   2e2j_2e2 2e2j2p1 2ad45 2ar0.785398
200j_200 628.319j6.28319 1.41421j1.41421 1.41421j1.41421

   16b1f 10b23 _10b23 1e2b23 2b111.111
31 23 _17 203 7.875
```

Отрицательные целые, после p и x соответствуют обратной степени. Например: 2p_2 означает два, деленное на π в квадрате, а 2x_2 есть два, деленное на квадрат числа Эйлера.

Управляющие Конструкции

Ключевые слова являются пунктуацией, определяющей порядок исполнения в Явных Определениях (`:`). Некоторые ключевые слова (включая охватываемые ими предложения) группируются в управляющие конструкции. Доступны следующие ключевые слова и управляющие конструкции:

```
assert. T

break.

continue.

for.     T do. B end.
for_xyz. T do. B end.

goto_name.
label_name.

if. T do. B end.

if. T do. B else. B1 end.

if.     T  do. B
elseif. T1 do. B1
elseif. T2 do. B2
end.

return.

select. T
 case.  T0 do. B0
 fcase. T1 do. B1
 case.  T2 do. B2
end.

throw.

try. B catch. B1 catchd. B2 catcht. B3 end.

while.  T do. B end.
whilst. T do. B end.
```

Начинающиеся на **B** или **T** слова, обозначают блоки, состоящие из нуля или более простых предложений и управляющих конструкций. В большинстве случаев, ход исполнения определяется результатом последнего предложения, выполненного в **T** блоке. Точнее, наличием ненулевого значения в его первом атоме (пустой результат **T** блока или опущенный **T** блок означают "истину"). Конечным результатом всей конструкции является результат последнего предложения, выполненного не в **T** блоке; если такого предложения не было,

конечным результатом является `i.0 0` .

Эти ключевые слова и управляющие конструкции детально описаны на следующих страницах.

assert.

```
assert. T
```

Приводит к выдаче "ошибки в предположениях" ("assertion failure"), если единственное предложение T не имеет результатом массив, состоящий только из единиц (1). Проверкой предположений можно управлять при помощи 9!:34 и 9!:35, если она отключена T не выполняется.

Например:

```
cfi=: 4 : 0 " 0                 y-ое сочетание элементов из i.x
  assert. 0<:y
  assert. y=<.y
  assert. y<2^x
  v=. +/\(i.x)!x
  m=. (y<v)i. 1
  (m,x) ci (y-m{0,v)
)

ci=: 4 : 0 " 1 0
 'm n'=. x
 if. 0=m do.
  i.0
 else.
  v=. +/\ (m-1)!(1-m)}.i.-n
  k=. (v>y) i. 1
  k,(1+k)+(x-1,1+k)ci(y-k{0,v)
 end.
)

   5 cfi 6
0 1

   5 cfi 6+i.10
0 1
0 2
0 3
0 4
1 2
1 3
1 4
2 3
2 4
3 4

   (i.100) -: 100x cfi <:2^100x
1
```

```
   5 cfi 33
|assertion failure: cfi
|       y<2^x
   5 cfi 6.2
|assertion failure: cfi
|       y=<.y
   5 cfi 'a'
|domain error: cfi
|   y=    <.y
```

break.

`break.` используется внутри управляющих конструкций `for.`, `while.`, или `whilst.` для перехода в конец самой внутренней из них. См. также `continue.`.

Например:

```
itn=: 3 : 0                    NB. Обратное треугольное число
 s=.0
 for_j.
  i.2e4
 do.
  if. s>:y do. j break. end.
  s=.j+s
 end.
)

   x=: 10 100 1000 10000
   itn"0 x
5 15 46 142

   ]y=: 2&!^:_1 x
5 14.651 45.2242 141.922

   2!y
10 100 1000 10000
```

continue.

`continue.` используется внутри управляющих конструкций `for.`, `while.`, or `whilst.` для перехода в начало самой внутренней из них. См. также `break.`.

Например:

```
sumeven=: 3 : 0               NB. Сумма четных
 s=.0
 for_j. i.y do.
  if. 2|j do. continue. end.
  s=.j+s
 end.
)

   sumeven 9
20

   +/ (* -.@(2&|)) i.9
20

   sumeven 1000
249500

   +/ (* -.@(2&|)) i.1000
249500

   (sumeven = 2&!@>.&.-:) 1000
1
```

for.

```
for.     T do. B end.
for_xyz. T do. B end.
```

Блок B выполняется по разу для каждого элемента массива A, являющегося результатом выполнения блока T. При использовании формы for_xyz., перед каждым выполнением блока B значению текущего элемента присваивается локальное имя xyz, а его индексу имя xyz_index (результаты непредсказуемы, если этим именам присваиваются другие значения внутри блока B).

break. заканчивает выполнение управляющей конструкции for., а continue. переходит к выполнению блока B для следующего элемента.

Например:

```
f0=: 3 : 0
 s=. 0
 for. i. y do. s=.>:s end.
)

   (f0 = ])"0 ?5$100
1 1 1 1 1

f1=: 3 : 0
 s=.0
 for_j. i.y do.
  if. 2|j do. continue. end.
  s=.j+s
 end.
)

   (f1 = 2&!@>.&.-:)"0 ?5$100
1 1 1 1 1

comb=: 4 : 0        NB. Все выборки длины x из i.y
 k=. i.>:d=.y-x
 z=. (d$<i.0 0),<i.1 0
 for. i.x do. z=. k ,.&.> ,&.>/\. >:&.> z end.
 ; z
)

   3 comb 5
0 1 2
0 1 3
0 1 4
0 2 3
```

```
0 2 4
0 3 4
1 2 3
1 2 4
1 3 4
2 3 4

queens=: 3 : 0      NB. Решает "задачу о N ферзях"
 z=.i.n,*n=.y
 for. }.z do.
  b=. -. (i.n) e."1 ,. z +"1 _ ((-i.){:$z) */ _1 0 1
  z=. ((+/"1 b)#z),.n|I.,b
 end.
)

   queens 5
0 2 4 1 3
0 3 1 4 2
1 3 0 2 4
1 4 2 0 3
2 0 3 1 4
2 4 1 3 0
3 0 2 4 1
3 1 4 2 0
4 1 3 0 2
4 2 0 3 1
```

goto_name.

`goto_name.` переходит к выполнению соответствующего `label_name.`. Эти ключевые слова включены для облегчения моделирования некоторых процессов.

Например:

```
f=: 3 : 0
 if. y do. goto_true. else. goto_false. end.
 label_true.  'true'  return.
 label_false. 'false' return.
)

   f 0
false

   f 1
true

   f ''
true
```

if.

```
if. T do. B end.
if. T do. B else. B1 end.
if. T do. B elseif. T1 do. elseif. T2 do. B2 end.
```

Результат последнего предложения в блоке T проверяется на ненулевое значение своего первого атома, определяя -- выполняется ли блок B после do. или остальная часть конструкции. Пустой результат блока T или отсутствие блока T означают "истину".

См. также конструкцию select. .

Например:

```
plus=: 4 : 0                    NB. Сложение для неотрицательных целых
 if. y do. >: x plus <: y else. x end.
)

   plus"0/~ i.5
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8

sel=: 1 : 'x # ['

quicksort=: 3 : 0
 if. 1 >: #y do. y
 else.
  (quicksort y <sel e),(y =sel e),quicksort y >sel e=.y{~?#y
 end.
)

   quicksort 15 2 9 10 4 0 13 13 18 7
0 2 4 7 9 10 13 13 15 18

perm=: 3 : 0                    NB. Все перестановки i.y
 if. *y do. ,/ (0 ,. perm&.<: y) {"2 1 \:"1 =i.y else. i.1 0 end.
)

   perm 3
0 1 2
0 2 1
1 0 2
1 2 0
2 0 1
2 1 0
```

```
test=: 3 : 0
 if.     0<y do. 'positive'
 elseif. 0>y do. 'negative'
 elseif.     do. 'zero'
 end.
)

   test&.> 5 _2.71828 0
+--------+--------+----+
|positive|negative|zero|
+--------+--------+----+

   test&.> '' ; 0 1 _2 ; _3 9
+--------+----+--------+
|positive|zero|negative|
+--------+----+--------+
```

return.

`return.` прекращает выполнение глагола, наречия или союза.

Результат явного определения -- результат последнего предложения, выполненного не в тестовом блоке (блоке T). Если такого предложения не было, результатом -- `i. 0 0` .

Например:

```
f=: 3 : 0
 try.
  if. 0<:y do. 'positive' return. else. t=.'negative' end.
 catch.
  t=.'caught'
 end.
 'it is ',t
)

   f 7
positive

   f _12
it is negative

   f 'deipnosophist'
it is caught

   (i.0 0) -: 3 : 'return.' 999
1

   g=: 3 : 'if. 13=|y do. ''triskaidekaphobia'' end.'

   g 13
triskaidekaphobia

   g 5
   (i.0 0) -: g 5
1
```

select.

```
select. T
 case.  T0 do. B0
 case.  T1 do. B1
 fcase. T2 do. B2
 case.  T3 do. B3
end.
```

Производится последовательное сравнение результата R блока T с результатами Ri блоков Ti , совпадение приводит к выполнению блока Bi соответствующего case. или fcase. . Если совпадение было в case. , выполнение конструкции select. заканчивается. Если в fcase. , выполняется следующий блок B(i+1) (и далее, если *он* тоже в fcase.).

Сравнение осуществляется как R e.&boxifopen Ri , где boxifopen=:<^:(L.=0:). Отсутствие Ti считается совпадением.

Например:

```
f0=: 3 : 0
 select. y
  case. 1;2 do. 'one two'
  case. 3   do. 'three'
  case. 4;5 do. 'four five'
  case. 6   do. 666
 end.
)

   f0&.> 1 2 3 4 5 6
+-------+-------+-----+---------+---------+---+
|one two|one two|three|four five|four five|666|
+-------+-------+-----+---------+---------+---+

   (i.0 0) -: f0 7
1

f1=: 3 : 0
 select. y
  case. 'a' do. i.1
  case. 'b' do. i.2
  case.     do. i.3
 end.
)

   f1&.> 'a' ; ('a';'b') ; 'b' ; 'x'
+-+-+---+-----+
|0|0|0 1|0 1 2|
+-+-+---+-----+
```

```
f2=: 3 : 0
 t=. ''
 select. y
  case.  1 do. t=.t,'one '
  fcase. 2 do. t=.t,'two '
  case.  3 do. t=.t,'three '
  fcase. 4 do. t=.t,'four '
 end.
)

   f2 1
one

   f2 2
two three

   f2 3
three

   f2 4
four

   '' -: f2 5
1
```

throw.

throw. переводит выполнение в секцию catcht. явного определения, приведшего к исполнению этого кода. Если соответствующая секция catcht. отсутствует, выходит в интерактивный режим.

Например:

```
main=: 3 : 0
 try.
  sub y
 catcht.
  select. type_jthrow_
   case. 'aaaa' do. 'throw aaaa'
   case. 'bbb'  do. 'throw bbb'
   case. 'cc'   do. 'throw cc'
   case.        do. throw.   NB. к catcht. следующего уровня (если есть)
  end.
 end.
)

sub=: 3 : 0
 if. y<0 do. type_jthrow_=: 'aaaa' throw. end.
 if. y<4 do. type_jthrow_=: 'bbb'  throw. end.
 if. y<8 do. type_jthrow_=: 'cc'   throw. end.
 (":y),' not thrown'
)

   main _4
throw aaaa
   main 1
throw bbb
   main 5
throw cc
   main 88
88 not thrown
```

Как иллюстрирует этот пример, throw. может передать информацию в catcht. блок при помощи глобального имени в некотором пространстве имен.

try.

```
try. B0 catch. B1 catchd. B2 catcht. B3 end.
```

Управляющая конструкция try/catch может содержать одно или более ключевых слов `catch.` `catchd.` `catcht.` , в любом порядке. Например:

```
try. B0 catch. B1 end.
try. B0 catcht. B1 catchd. B2 end.
try. B0 catcht. B1 catch. B2 catchd. B3 end.
```

Исполняет блок `B0`, и:
• `catch.` перехватывает ошибку в `B0` , невзирая на значение флага отладки `13!:0`
• `catchd.` перехватывает ошибку в `B0` , но только если флаг отладки установлен в 0
• `catcht.` перехватывает `throw.` в явном определении, вызванном из `B0`

Например:

```
f=: 4 : 0
 try.
  try. 3+y catch. *:x end.
 catch.
  'x and y are both bad'
 end.
)

   13 f 7
10

   13 f 'primogeniture'
169

   'sui' f 'generis'
x and y are both bad
```

while.

```
while.  T do. B end.
whilst. T do. B end.
```

Результат последнего, выполненного в блоке T , предложения проверяется на ненулевое значение своего первого атома. Если тест дает "истину", выполняется блок B . Потом блок T выполняется снова... и так далее, пока очередная проверка результата блока T не дает "ложь". (Пустой результат блока T , или опущенный блок T соответствуют "истине".)

`whilst.` отличается от `while.` только тем, что пропускает тест (**s**kips **t**est) перед первым выполнением блока B.

`break.` заканчивает выполнение управляющей конструкции `while.` или `whilst.`, а `continue.` переводит выполнение к ее началу.

Например:

Возведение в целую степень квадратами

```
exp =: 4 : 0
 z=.1
 a=.x
 n=.y
 while. n do.
  if. 2|n do. z=.z*a end.
  a=.*:a
  n=.<.-:n
 end.
 z
)

   3 exp 7
2187

   3 ^ 7
2187

   1.1 exp 0
1

   2x exp 128
340282366920938463463374607431768211456
```

Ссылки

1. Falkoff, A.D., and K.E. Iverson, *The Design of APL*, IBM Journal of Research and Development, July, 1973.

2. Falkoff, A.D., and K.E. Iverson, *The Evolution of APL*, ACM Sigplan Notices, August 1978.

3. Iverson, K.E., *A Dictionary of APL*, ACM APL Quote-Quad, September, 1987.

4. McIntyre, D.B., *Language as an Intellectual Tool: From Hieroglyphics to APL*, IBM Systems Journal, December, 1991.

5. Iverson, K.E., *A Personal View of APL*, IBM Systems Journal, December, 1991.

6. Hui, R.K.W., *An Implementation of J*, ISI, 1992.

7. Reiter, C.A., *Fractals, Visualization, and J*, ISI, 1995.

8. Iverson, K.E., *Exploring Math*, ISI, 1996.

9. Burke, C. et al., *J Phrases*, ISI, 1998.

10. McDonnell, E.E., *Complex Floor*, APL Congress 73, North-Holland/American Elsevier.

11. McDonnell, E.E., *Zero Divided by Zero*, APL76, ACM.

12. Bernecky, Robert, and R.K.W. Hui, *Gerunds and Representations*, APL91, ACM.

13. Abramowitz, M., and I.A. Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards Applied Mathematics Series #55, U. S. Government Printing Office, 1964.

14. Iverson, K.E., *Concrete Math Companion*, ISI, 1995.

15. Knuth, D.E., *The Stanford GraphBase*, Addison-Wesley, 1994.

Благодарности

Мы благодарим Эрика Айверсона (Eric Iverson) за проэктирование и реализацию локативов, управляющих конструкций и пользовательского интерфейса для операционной системы Windows.

Константин Л. Метлов благодарит Алекса Корниловского, Бойко Банчева, и других за помощь в подборе и обсуждении русских терминов, использованных в данном переводе.

Приложение A. Внешний (союз)

В применении к целым скалярным левому и правому аргументу, союз `!:` производит глаголы (кроме случая `5!:0` , производящего наречие). Эти глаголы ведут себя как любые другие: им можно присвоить имена, они могут служить аргументами для наречий или союзов, их нужно вызывать с аргументом, даже если его значение не играет роли (как, например, пустая строка в случае `7!:0 ''`). Имена, передаваемые в качестве аргументов этих глаголов, должны быть упакованы (как в случае `4!:55 'a';'bc'` для стирания имен `a` и `bc`). Если в дальнейшем описании левый аргумент заключен в квадратные скобки, то его можно опустить.

`0!:k y` Сценарий y выполнятся в соответствии с тремя цифрами (ноль или единица) в десятичном представлении k :

	1-я цифра	2-я цифра	3-я цифра
0	Из файла или сущ.	Прекратить при ошибке	Без вывода
1	Из существительного	Продолжить при ошибке	Отображать выполнение

Например, `0!:111 abc` выполняет *существительное* `abc` *до конца* и *показывает выполняемые предложения*.

Сессии интерпретатора начинаются с выполнения (без вывода, с остановкой при ошибке) `0!:0<'profile.ijs'`

`0!:2 y` Сценарий y считается последовательностью тавтологий; `0!:2 y` работает как `0!:1 y`, но останавливается, если какое-либо не присвоенное существительное-результат очередного выполненного предложения не состоит из одних только единиц.

`0!:3 y` Как `0!:2 y` , но производит результат `1` или `0` в соответствии с тем -- "проходит" ли сценарий (содержит только тавтологии) или нет.

Файлы 1!:

Если не указано иначе, на файл можно сослаться по его имени (как `<'sub\abc.q'`) или при помощи целого числа (номера файла), полученного при его открытии (`1!:21 <'sub\abc.q'`).

`1!:0 y` **Каталог**. y -- строка поиска по файловой системе (упакованная строка тоже принимается); результат -- таблица из 5-ти столбцов, содержащая упакованные: имя файла, время последнего изменения, размер, права доступа и аттрибуты. Например, попробуте выполнить `1!:0 '*.*'`. Права доступа и аттрибуты зависят от операционной системы. Например, в Windows:

```
   1!:0 'j.exe'
+-----+----------------+------+---+------+
|j.exe|1998 2 2 14 33 46|676864|rwx|-----a|
+-----+----------------+------+---+------+
```

Права доступа -- строка из трех букв, соответствующих разрешениям читать, записывать и выполнять файл, соответственно. Аттрибуты -- строка из 6-ти символов, соответствующая аттрибутам: "только для чтения", "скрытый", "системный", "метка тома", "каталог" и "требует архивирования".

`1!:1 y` **Прочитать**. y -- имя файла или его номер (полученный через `1!:21`); результат -- содержимое файла, представленное в виде строки. Например, `1!:1 <'abc.q'`. Допустимы также следующие значения y :

1 читать с клавиатуры (не работает в сценариях)
3 читать из стандартного входного потока (stdin)

`x 1!:2 y` **Записать**. x -- строка, которая должна стать новым содержимым файла; y -- имя файла или его номер (полученный через `1!:21`). Допустимы также следующие значения для y :

2 вывод на терминал.
4 стандартный выходной поток (stdout)
5 стандартный поток ошибок (stderr)

`x 1!:3 y`

Добавить. Как `x 1!:2 y`, но добавляет к текущему содержимому файла, а не заменяет его.

`1!:4 y` **Размер**.

`1!:5 y` **Создать Каталог**. `y` -- (упакованное) имя каталога

`[x] 1!:6 y` **Запросить/Установить Аттрибуты**.

`[x] 1!:7 y` **Запросить/Установить Права Доступа**.

`1!:11 y` **Прочитать по Индексу**. `y` -- список из упакованного имени (номера) файла и упакованных вместе индекса и длины. Индекс может быть отрицательным. Если длина опущена, чтение производится до конца файла. Например:

```
1!:11 'abc.x';1000 20
f=: 1!:21 <'abc.x'
1!:11 f,1000 20
```

`x 1!:12 y` **Записать по Индексу**. `x` -- строка, которая должна быть записана; `y` -- список из упакованного имени (номера) файла и упакованного индекса.

`1!:20 y` **Номера и Имена Файлов**. Таблица из 2-х столбцов, где перечислены номера и имена открытых файлов.

`1!:21 y` **Открыть**. Открыть Файл по имени `y`, создав его, если требуется; результат -- номер файла.

`1!:22 y` **Закрыть**. Закрыть Файл по имени (номеру) `y`. Все блокировки снимаются.

`1!:30 y` **Блокировки**. Таблица из 3-х целых столбцов: номер файла, индекс и длина блокированного участка в файле. Аргумент `y` нужен, но игнорируется.

`1!:31 y` **Блокировать**. `y` -- целый вектор из 3-х элементов: номер файла, индекс и длина участка в файле, который требуется заблокировать; результат -- 1 (если запрос выполнен успешно) или 0 (если нет).

`1!:32 y` **Разблокировать**. `y` -- целый вектор из 3-х элементов: номер файла, индекс и длина участка в файле, который

требуется разблокировать.

`1!:43 y` **Запросить Текущий Каталог**. Запросить рабочий каталог (Posix `getcwd`).

`1!:44 y` **Установить Текущий Каталог**. Установить рабочий каталог (Posix `chdir`).

`1!:46 y` **Запросить путь к DLL**. Возвращает путь (в формате UTF8) к j.dll в системе Windows и `''` в Unix.

`1!:55 y` **Стереть Файл/Каталог**. Например, `1!:55<'careful'`

Система 2!:

`2!:0 y` **Выполнить Команду**. Список y передается операционной системе для исполнения, возвращая результат. Например, `2!:0 'dir *.exe'`. Не доступно под Windows.

`2!:1 y` **Создать Процесс**. (только под Unix.) Как `2!:0`, но возвращает `''` не дожидаясь завершения команды. Вывод игнорируется. Например, `2!:1` можно использовать для вызова текстового редактора.

`2!:2 y` **Прицепить Процесс**. (Только под Unix.) Командная строка y передается на обработку `/bin/sh` , присоединяя два номера файлов к стандартному входному и выходному потокам команды. Результат -- список из 3-х элементов: идентификатор запущенного процесса и номера файлов его стандартных входного и выходного потоков. Эти номера файлов также появляются в среди результатов `1!:20` , только вместо имени файла в таблице указывается командная строка с префиксами > (для входного потока) или < (для выходного потока). Файлы, связанные с процессом, должны быть закрыты глаголом `1!:22` , когда они больше не нужны. См. также `2!:3` для глагола, при помощи которого можно дождаться окончания выполнения процесса.

`2!:3 y` **Ждать**. (Толко под Unix.) Ждать окончания процесса с идентификатором y . Результат -- код окончания, возвращенный окончившимся процессом.

`2!:5 y` **Getenv**. Значение переменной окружения по имени y. Если такая переменная не определена, результат -- 0.

`2!:6 y` **Getpid**. Идентификатор процесса.

`2!:55 y` **Закончить Сессию**. y -- код окончания, целое число.

Преобразования 3!:

`3!:0 y` **Тип**. Внутренний тип существительного `y`, кодированный следующим образом:

		1024	разреженный булевский
1	булевский	2048	разреженный текстовый
2	текстовый	4096	разреженный целый
4	целое	8192	разреженный действительный
8	действительное	16384	разреженный комплексный
16	комплексное	32768	разреженный упакованный
32	упаковка	65536	символ
64	целое произв. точн.	131072	Уникод (Unicode)
128	рациональное		

`[x] 3!:1 y` **Преобразовать в Машинное Представление**. В *стандартном порядке* (little endian), байты в слове упорядочены от наиболее значимого к менее значимому; в *обратном порядке* (big endian), байты упорядочены от менее значимого к более значимому. Например, 4-х байтовое целое 265358979 представляется как `0fd10e83` в стандартном порядке и как `830ed10f` в обратном порядке. В PC используется обратный порядок (big endian).

В применении к массиву `y`, диада `x 3!:1 y` производит его машинное представление в соответствии со значением атома `x`:

x	длина слова	порядок байт
0	32 бита	стандартный
1	32 бита	обратный
2	64 бита	стандартный
3	64 бита	обратный

В качестве значения аргумента `x` можно использовать 10 вместо 2 и 11 вместо 3 для

совместимости с предыдущими версиями. Монада `3!:1` производит машинное представление для длины слова и упорядочения байт машины, на которой она выполняется.

Результат `3!:1 y` или `x 3!:1 y` состоит из следующих частей. Где mword (машинное слово) имеет длину 32 или 64 бита, как определено параметром `x` или, если `x` опущен, машиной, на которой выполняется программа.

флаг	1 mword	Первый байт кодирует одно из следующего: `e0` 32 бита, стандартный `e1` 32 бита, обратный `e2` 64 бита, стандартный `e3` 64 бита, обратный
тип	1 mword	как в `3!:0`
#элементов	1 mword	`*/$y` для плотного массива; `1` для разреженного
ранг	1 mword	`#$y`
размерность	rank mwords	`$y`
значение	? mwords	элементы разобранного `y`

Для элементов, представляющих собой упаковки, целые произвольной точности, рациональные и прореженные типы записываются смещения в байтах представлений соответствующих элементов.

См. `3!:3` ниже для примеров. `3!:2 y` **Преобразовать из Двоичного/Шестнадцатеричного Представления**. Обратный глагол к `3!:1` и `3!:3` ; работает с аргументами любой длины слова и любого упорядочения байт. Если первый байт флагов (см. `3!:1` выше) не представляет собой `e0 e1 e2 e3` , тогда преобразование

производится в предположении, что данные записаны J версии до 6.01.

`[x] 3!:3 y` **Шестнадцатеричное Представление**. Как `3!:1` , но результатом является текстовая матрица в шестнадцатеричном представлении. Например под 32-битной Windows:

```
   (3!:3 x) ; (3!:3 x,o.1) ; 2 (3!:3) x,o.1 [ x=: 1 2 3 0 _1
+--------+--------+----------------+
|e1000000|e1000000|e200000000000000|
|04000000|08000000|0000000000000008|
|05000000|06000000|0000000000000006|
|01000000|01000000|0000000000000001|
|05000000|06000000|0000000000000006|
|01000000|00000000|3ff0000000000000|
|02000000|0000f03f|4000000000000000|
|03000000|00000000|4008000000000000|
|00000000|00000040|0000000000000000|
|ffffffff|00000000|bff0000000000000|
|        |00000840|400921fb54442d18|
|        |00000000|                |
|        |00000000|                |
|        |00000000|                |
|        |0000f0bf|                |
|        |182d4454|                |
|        |fb210940|                |
+--------+--------+----------------+

   t=: 0 (3!:3) ;:'fourscore and ten years ago'
   $t
43 8
   12{.t
e0000000
00000020
00000005
00000001
00000005
00000028
00000048
00000060
00000078
00000094
e0000000
00000002

   dfh=: 16 #. '0123456789abcdef' i. ]  NB. из шестнадцатеричного

   ((i.#t) e. 0,4 %~ dfh (5+i.5){t)
```

`3!:4 y`
`3!:5 y` **Преобразование между Целыми/Действительными**. Если `ic=: 3!:4` и `fc=: 3!:5` , тогда

`3 ic y`	целые языка J в наборы из 8-ми байт (только в J64)
`_3 ic y`	наборы 8-ми байт в целые языка J (только в J64)
`2 ic y`	целые языка J в наборы 4-х байт
`_2 ic y`	наборы 4-х байт в целые J
`1 ic y`	целые J в наборы 2-х байт
`_1 ic y`	наборы 2-х байт в целые J
`0 ic y`	наборы 2-х байт, интерпретируемые без знака в целые J
`2 fc y`	действительные J в машинные с плавающей точкой (double)
`_2 fc y`	машинные с плавающей точкой (double) в действительные J
`1 fc y`	действительные J в машинные короткие с плавающей точкой (float)
`_1 fc y`	машинные короткие с плавающей точкой (float) в действительные J

Все ранги равны бесконечности, обратные глаголы к `k&ic` и `k&fc` определены для всех `k`, кроме `k=0`. `3!:6 y`**Блокировать Сценарий**. Кодирует текст сценария, превращая его в блокированный сценарий.

`4!:0 y` **Класс Имени**. Класс (упакованного) имени:

`_2`	не имя
`_1`	не существует
`0`	существительное
`1`	наречие
`2`	союз
`3`	глагол

`[x] 4!:1 y` **Список Имен**. Результат -- вектор упакованных имен, принадлежащих к классам от `0` до `3` (как определено для `4!:0` выше) либо названий пространств имен, если аргумент равен `6`. Необязательный левый аргумент позволяет указать первые буквы имен, которые должны быть включены.

`4!:3 y` **Сценарии**. Список имен сценариев, которые были выполнены при помощи `0!:n`

`4!:4 y` **Индекс Сценария**. Индекс (в `4!:3 ''`) сценария, который определил `y` , или `_1` если имя `y` не было определено в сценарии.

`4!:5 y` **Измененные Имена**. `4!:5]0` выключает сбор данных; `4!:5]1` включает его и производит список глобальных имен, значение которых изменилось от времени последнего выполнения `4!:5` .

`4!:55 y` **Стереть**.

Представление 5!:

`x 5!:0` **Определить**. `5!:0` является наречием, которое полностью обращает действие `5!:1`. Тоесть, `(5!:1 <'f') 5!:0` равно `f` для всех `f` .

`5!:1 y` **Атомное**. Атомное представление обьекта по имени `y` , используется для построения герундиев. Результат -- упаковка, содержащая либо строку, представляющую обьект, если он является примитивом; либо двух-элементный упаковочный список из символа и атомного представления его аргументов, если обьект примитивом не является. Бессимвольные части речи кодируются следующим образом:

`0`	Существительное
`2`	Крючок
`3`	Вилка
`4`	Союз с прицепом или цепочка наречий

Например:

```
   plus=: +
   5!:1 <'plus'
+-+
|+|
+-+
   noun=: 3 1 4 1 5 9
   5!:1 <'noun'
+---------------+
|+-+-----------+|
||0|3 1 4 1 5 9||
|+-+-----------+|
+---------------+
   increment=: 1&+
   5!:1 <'increment'
+-------------+
|+-+---------+|
||&|+-----+-+||
|| ||+-+-+|+|||
|| |||0|1|| |||
|| ||+-+-+| |||
|| |+-----+-+||
|+-+---------+|
+-------------+
```

`5!:2 y` **Коробочное**.

```
   nub=: (i.@# = i.~) # ]
   5!:2 <'nub'
```

```
+------------------+-+-+
|+--------+-+------+|#|]|
||+--+-+-+|=|+--+-+|| | |
|||i.|@|#|| ||i.|~||| | |
||+--+-+-+| |+--+-+|| | |
|+--------+-+------+| | |
+------------------+-+-+
```

5!:4 y **Древовидное**. Текстовая матрица, представляющая именованную часть речи в форме дерева. Таким образом:

```
   5!:4 <'nub'
            +- i.
      +- @ -+- #
  +---+- =
  |   +- ~ --- i.
--+- #
  +- ]
```

5!:5 y **Линейное**. Линейное представление -- строка, которая, будучи интерпретированной, воспроизводит именованный обьект. Например:

```
   5!:5 <'nub'
(i.@# = i.~) # ]

   5!:5 <'a' [ a=: o. i. 3 4
3.14159265358979324*i.3 4

   lr=: 3 : '5!:5 <''y'''
   lr 10000$'x'
10000$'x'
```

5!:6 y **Скобочное**. Как и линейное представление, но порядок выполнения указан явно при помощи скобок.

```
   5!:6 <'nub'
((i.@#) = (i.~)) # ]
```

x 5!:7 y **Явное**. Левый аргумент 1 (для монадного случая) или 2 (для диадного); правый аргумент -- упакованное имя глагола, наречия или союза. Например:

```
   perm=: 3 : 0
    z=. i.1 0
    for. i.y do. z=.,/(0,.1+z){"2 1\:"1=i.>:{:$z end.
   )

   1 (5!:7) <'perm'
+-+----------+------------------------------+
|0|1 _1 0    |z=.i.1 0                      |
+-+----------+------------------------------+
|1|65536 2 1 |for.                          |
+-+----------+------------------------------+
|2|2 _1 1    |i.y                           |
```

```
+-+----------+------------------------------+
|3|131072 6 1|do.                           |
+-+----------+------------------------------+
|4|1 _1 1    |z=.,/(0,.1+z){"2 1\:"1=i.>:{:$z|
+-+----------+------------------------------+
|5|32 3 1    |end.                          |
+-+----------+------------------------------+
```

Результат `5!:7` представляет из себя упакованную матрицу из 3-х столбцов. В столбце 0 находятся упакованные целые `0 1 2 ... n-1`. В столбце 1 упакованные 3-х элементные целые векторы управляющей информации: код ключевого слова, номер следующей строки, номер строки в исходном определении. В столбце 2 упакованные ключевые слова и предложения.

Если именованый обьект не определен явно или для указанной валентности, результат `5!:7` -- пустая матрица размерности `0 3`, .

Для всех частей речи, кроме существительных, представление по умолчанию, устанавливаемое при помощи `9!:3` , дает возможность экспериментировать со всеми представлениями. Например, `9!:3 (6 4 2)` заставляет интерпретатор показывать и скобочное, и древовидное, и коробочное представления.

Время 6!:

См. также 9!:32 and 9!:33 .

6!:0 y **Сейчас**. Текущее время в порядке: год, месяц, день, час, минуты, секунды. Аргументом может быть пустой вектор, тогда глагол возвращает 6-ти элементный числовой список; или строка из букв `Y M D h m s` и других, тогда глагол форматирует время в виде строки. Например:

```
   6!:0 ''
2007 10 30 13 10 45.312

   6!:0 'YYYY-MM-DD hh:mm:ss.sss'
2007-10-30 13:10:46.875

   6!:0 'hh:mm:ss MM/DD/YY'
13:10:48 10/30/07
```

6!:1 y **За Сессию**. Секунды с начала сессии

[x] 6!:2 y **На Выполнение**. Время, потребовавшееся для выполнения предложения y , в секундах. Среднее x-кратного выполнения, по умолчанию однократного. Например:

```
   a=:?50 50$100
   6!:2 '%.a'
0.091
   10 (6!:2) '%.a'          NB. Среднее 10-ти выполнений
0.0771
   ts=: 6!:2 , 7!:2@]       NB. Время и использ. память
   ts '%.a'
0.08 369920
```

6!:3 y **Задержка**. Остановиться на y секунд. Например, `6!:3 (2.5)`

6!:4 y **Вызовы парсера**. Количество обращений к парсеру J с начала сессии. Счетчик представляет собой машинное целое, его инкремент осуществляется без проверки на переполнение.

Например:

```
   6!:4 '' NB. 0
2081
   t=:3+5
   6!:4 '' NB. 1
```

```
2083
   (3 : 'z=.0 for. i.y do. z=.>:z end. *:z') 100
10000
   6!:4 '' NB. 2
2188
   $".100$,:'0+1'
100
   6!:4 '' NB. 3
2290
```

Между `NB. 0` и `NB. 1` , парсер был вызван дважды: один раз для `t=:3+5` и второй раз для `6!:4 '' NB. 1` .

Между `NB. 1` и `NB. 2` , парсер был вызван 105 раз:

```
  1   (3 : 'z=.0 for. i.y do. z=.>:z end. *:z')
      100
  1   z=.0
  1   i.y
100   z=.>:z
  1   *:z
  1   6!:4 '' NB. 2
```

Между `NB. 2` и `NB. 3` , парсер был вызван 102 раза: первый раз для `$".100$,:'0+1'` , потом 100 раз для каждого выполнения `0+1` , и, наконец, для `6!:4 '' NB. 3` .

`6!:8 y` **Частота Часов**. Возвращает частоту часов, количество отсчетов в секунду. (Под Windows, запрашивается функция `QueryPerformanceFrequency`.) Аргумент `y` игнорируется.

```
   6!:8 ''
1.19318e6
```

`6!:9 y` **Количество Отсчетов**. Возвращает текущее количество отсчетов в часах (прошедшее время). (В Windows, запрашивается `QueryPerformanceCounter`.) Аргумент `y` игнорируется.

```
   6!:9 ''
4.08486e9
   _10 [\ 2 -~/\ 6!:9"1 (101 0)$0
9 7 7 6 7 5 7 6 6 7
6 7 6 6 6 6 8 6 6 7
6 6 7 6 6 7 6 6 6 6
6 7 6 6 6 6 7 6 6 8
6 7 6 6 6 7 6 6 6 7
6 6 6 7 7 6 7 6 7 6
6 6 6 7 6 6 6 6 7 6
```

```
6 6 6 7 6 7 7 6 7 6
6 6 6 6 7 6 6 6 7 6
6 6 6 6 6 6 6 6 6 8
```

`[x] 6!:10 y` **Область Данных Монитора Производительности**. Текстовый вектор y отображается на фиксированную структуру записей "упакованных" данных монитора производительности (МП); старая область данных МП (если она была определена) освобождается. Счетчик МП (см. `6!:12`) устанавливается в 0. Если y пустой, данные собраны не будут.

x -- 2-х элементный вектор управляющих параметров; по умолчанию (если x опущено) предполагается равным `0 0`.

```
0{x - что записывать
        0 - вход и выход
        1 - вход и выход, и все строки
1{x - что делать при нехватке места в y
        0 - "обернуть", отбросить самую старую запись
        1 - "обрезать", отбросить новую запись
```

Результат -- количество записей, которое может поместиться в y.

`6!:11 y` **Распаковать данные МП**. Распаковать данные МП, в упакованый вектор из следующих колонок:

```
0  имя
1  пространство имен, в котором оно определено
2  валентность
3  номер строки (_1 на входе; _2 на выходе)
4  используемая память (в байтах)
5  время (в секундах)
6  список имен
```

Столбцы 0 и 1 являются индексами в содержимое столбца 6. Столбцы от 0 до 5 содержат одинаковое количество (взаимно соответствующих) элементов.

y может содержать индексы записей, которые требуется распаковать, где 0 -- самая старая запись (и `_1` самая новая запись). Если y пуст, распаковываются все записи. Строки в результате всегда расположены от более старых к более новым, ни одна строка не повторяется.

`6!:12 y`

Добавить y к счетчику МП. Данные МП записываются если счетчик МП больше нуля и определена область данных МП. (Непустая область данных МП должна быть определена перед установкой счетчика МП) Результат -- значение счетчика МП после операции.

`6!:13 y` **Статистика МП**. Производит 6-ти элементный вектор со следующей информацией:

0 - что записывать
    0 - вход и выход
    1 - вход, выход и все строки
1 - действие при нехватке памяти
    0 - обернуть
    1 - обрезать
2 - максимальоне кол-во записей
3 - текущее количество записей
4 - были-ли потеряны записи
5 - текущее значение счетчика МП

Если область данных МП не была определена, результат -- `6$0`. Аргумент y игнорируется.

Дальнейшие примеры:

```
   sum=: +/

   avg=: 3 : 0
n=. #y
s=. sum y
s % n
)

   1 (6!:10) 1e5$'x'
3570
   (6!:12) 1
1
   avg"1 ? 3 10$100
65 64.4 64.3
   (6!:12) _1
0

   x=. 6!:11 ''
   $&.> x
+--+--+--+--+--+--+-+
|21|21|21|21|21|21|3|
+--+--+--+--+--+--+-+
   |: > 6{. x
0 2 1 _1 1152 22582.5
```

```
0 2 1  0 4224 22582.5
0 2 1  1  576 22582.5
1 2 1 _1  128 22582.5
1 2 1 _2   64 22582.5
0 2 1  2  128 22582.5
0 2 1 _2    0 22582.5
0 2 1 _1  128 22582.5
0 2 1  0 4224 22582.5
0 2 1  1  576 22582.5
1 2 1 _1  128 22582.5
1 2 1 _2   64 22582.5
0 2 1  2  128 22582.5
0 2 1 _2    0 22582.5
0 2 1 _1    0 22582.5
0 2 1  0 4224 22582.5
0 2 1  1  576 22582.5
1 2 1 _1  128 22582.5
1 2 1 _2   64 22582.5
0 2 1  2  128 22582.5
0 2 1 _2    0 22582.5

   _5[\ 2 - ~/\ > 5{x
4.52571e_5 2.09524e_5 1.50857e_5 1.67619e_5 9.21905e_6
6.28571e_5 9.21905e_6 1.59238e_5 1.59238e_5 1.34095e_5
1.00571e_5 1.00571e_5 5.61524e_5 7.54286e_6 1.59238e_5
1.50857e_5 1.42476e_5 1.00571e_5 8.38095e_6 5.61524e_5

   > {: x
+---+---+----+
|avg|sum|base|
+---+---+----+

   (6!:10) 1e5$'x'   NB. только вход и выход
3570
   (6!:12) 1
1
   avg"1 ? 3 10$100
38.8 61.9 61.7
   (6!:12) _1
0

   x=. 6!:11 ''
   |: > 6{.x
0 2 1 _1  832 22681.5
1 2 1 _1 4928 22681.5
1 2 1 _2   64 22681.5
0 2 1 _2  128 22681.5
0 2 1 _1  128 22681.5
1 2 1 _1 4928 22681.5
1 2 1 _2   64 22681.5
0 2 1 _2  128 22681.5
0 2 1 _1    0 22681.5
1 2 1 _1 4928 22681.5
1 2 1 _2   64 22681.5
0 2 1 _2  128 22681.5

   _5[\ 2 - ~/\ > 5{x
```

```
7.12381e_5  1.50857e_5 6.70476e_5 9.21905e_6    3.52e_5
1.00571e_5 0.000108952 7.54286e_6 3.93905e_5 1.08952e_5
6.03428e_5           0          0          0          0

   6!:13 ''
0 0 3570 12 0 0

   sum_ab_ =: +/

   sum_z_ =: sum_ab_

   avg_q_ =: 3 : 0
tally=. #
s=. sum y
n=. tally y
s % n
)

   (6!:10) 1e5$'x'
3570
   (6!:12) 1
1
   avg_q_ i.12
5.5

   (6!:12) _1
0

   x=: 6!:11 ''
   (>{&>/0 6{x) ,. ' ' ,. (>{&>/1 6{x) ,. ' ' ,. ": ,. >3{x
avg_q_  q  _1
sum     q  _1
sum_ab_ ab _1
sum_ab_ ab _2
sum     q  _2
tally   q  _1
tally   q  _2
avg_q_  q  _2

   > 6{x
+------+---+-------+-----+-+--+
|avg_q_|sum|sum_ab_|tally|q|ab|
+------+---+-------+-----+-+--+
```

Память 7!:

См. также 9!:20 и 9!:21 .

7!:0 y **Сейчас**. Количество памяти, используемое в данный момент.

7!:2 y **На Выполнение**. Память, требуемая на выполнение предложения y. Например:

```
   7!:2 ',/x' [ x=: 3000 2 5$'kerygmatic'
33888
```

7!:3 y **Свободная**. Информация о состоянии подсистемы управления памятью J, сейчас таблица из двух столбцов: размеров и количества свободных блоков данного размера. Например:

```
   7!:3 ''
   32  950
   64 1135
  128  554
  256  157
  512   74
 1024   40
```

Определение и доступность глагола 7!:3 могут измениться.

7!:5 y **Занимает**. Обьем памяти в байтах, используемый для хранения обьектов (существительных, глаголов, наречий и союзов), перечисленных в y . Например:

```
   Hilbert=: % @: >: @: (+/~) @: i.

   7!:5 <'Hilbert'
896
   7!:5 <'name_without_value'
|value error: name_without_value
|       7!:5<'name_without_value'

   sp=: 3 : '7!:5 <''y'''
   sp 'chiaroscuro'
64
   sp i.1000
4096
   sp o.i.1000
8192
```

```
   x=: o.y=: i.1000
   7!:5 ;:'x y Hilbert'
8192 4096 896
```

`7!:6 y` **В Пространстве Имен**. Обьем памяти в байтах, занятый пространствами имен, перечисленными в y (включая память занятую именами, значениями этих имен в указанном пространстве, именем самого пространства, хеш таблицей и указателями на данное пространство в глобальной таблице имен). Например:

```
   7!:6 ;:'base z 0'
8256 198880 2168
   7!:6 <'12345'
|locale error
|       7!:6<'12345'
   7!:6 <'nonexistent'
|locale error
|       7!:6<'nonexistent'
```

Формат 8!:

`[x] 8!:0 y` **Форматировать**. Форматировать массив `y` в соответствии с форматом в `x` . Упаковать каждое отформатированное число.

`[x] 8!:1 y` **Форматировать1**. Как и `x 8!:0 y` , но `y` не может иметь ранг более 2-х, в окончательном результате упаковываются только столбцы.

`[x] 8!:2 y` **Форматировать2**. Как и `x 8!:0 y` , но `y` не может иметь ранг более 2-х, результат -- просто текстовая таблица.

Ранги `8!:n` равны бесконечности. `x 8!:n y` форматирует массив `y` в соответствии с форматом в `x` .

Обычно `y` -- массив действительных чисел. Но `y` может быть и текстовой таблицей, в этом случае `x` игнорируется, а результатом является `<@,"1 y` (с дальнейшей обработкой, в зависимости от варианта форматирующего глагола). Наконец, `y` может быть массивом упаковок, где каждый элемент (после распаковки) -- действительное число или строка; в этом случае, каждый распакованный атом форматируется независимо.

`x` -- строка фраз, отделенных запятыми; либо атом; либо список индивидуально упакованных фраз. Он должен содержать либо одну фразу (применяемую, ко всем столбцам) либо одну фразу на столбец. Если `x` опущен, вместо него берется `a:$~(_1+#$y)}.$y` . Тоесть, `8!:n y` эквивалентно вычислению числа позиций после десятичной точки независимо для каждой колонки, и использованию минимальной ширины для каждого числа.

Фраза формата состоит из нуля или более модификаторов, и необязательного суффикса `w.d` , устанавливающего ширину и количество цифр после десятичной точки. `w` может быть 0, и `w.` или `w.d` целиком можно опустить. Значение каждого из этих случаев следующее:

	ширина	цифры после точки
`w.d`	`w`	`d`

0.d	вычисляется	d
d	минимально	d
опущено	минимально	вычисляется

Указанная или вычисленная величина w применяется ко всему массиву, в то время как минимальное w применяется к каждому отдельному числу. Недостаточная ширина дает результат состоящий из *.

d не может быть более 9-ти. Вычисленное d применяется к целому массиву, и, в этом случае, для ненулевых чисел с абсолютной величиной менее 1e_9 или более 2e9 используется экспоненциальная (научная) запись.

Допустимы следующие модификаторы (каждый из них можно применить не более одного раза в одной фразе формата) :

c	вставить запятую между тройками цифр слева от десятичной точки
l	выровнять влево (w должно быть указано или вычислено)
b<xx>	заменитель для форматированного нуля
d<xx>	заменитель для _ __ _. ; при отсутствии d , _ __ _. показываются как есть
m<xx>	префикс для форматированных отрицательных чисел, заменяющий знак "минус" -
n<xx>	суффикс для форматированных отрицательных чисел
p<xx>	префикс для форматированных неотрицательных чисел
q<xx>	суффикс для форматированных неотрицательных чисел
r<xx>	фоновый текст, повторяется циклически как в диаде $
s<xx>	xx массив символов четной длины (по умолчанию,замена) указывающих на замену символов по умолчанию заменить можно символы: e,.-* знак "минус" по умолчанию выводится как - (вместо _)

Можно опустить <xx>, что означает <> . Например, просто b само по себе обозначает: "опустить нули, вместо них не выводить ничего". Вместо <xx> можно использовать (xx) или [xx] или {xx} .

Примеры:

```
   fmt =: 8!:0
   fmt1=: 8!:1
   fmt2=: 8!:2

   ] y=: 1.23 12345 123.4 0.12 ,__ 0 1.15 _1234.5,: _44 0.5 _0.5 0.1
1.23 12345 123.4    0.12
  __       0  1.15 _1234.5
 _44   0.5  _0.5     0.1
```

```
   '' fmt y            NB. минимальное w вычисленное d, одинаковое для всех столб
+------+--------+------+--------+
|1.23  |12345.00|123.40|0.12    |
+------+--------+------+--------+
|__    |0.00    |1.15  |-1234.50|
+------+--------+------+--------+
|-44.00|0.50    |-0.50 |0.10    |
+------+--------+------+--------+
   $ '' fmt y
3 4
   #&> '' fmt y
4 8 6 4
2 4 4 8
6 4 5 4

   ',,,' fmt y         NB. минимальное w вычисленное d, для каждого столбца индив
+------+-------+------+--------+
|1.23  |12345.0|123.40|0.12    |
+------+-------+------+--------+
|__    |0.0    |1.15  |-1234.50|
+------+-------+------+--------+
|-44.00|0.5    |-0.50 |0.10    |
+------+-------+------+--------+
   ((4$<'') fmt y) -: ',,,' fmt y
1
   (fmt y) -: ',,,' fmt y
1

   '0.2' fmt y         NB. вычисленное w, 2 знака после точки, для всех столбцов
+--------+--------+--------+--------+
|    1.23|12345.00|  123.40|    0.12|
+--------+--------+--------+--------+
|      __|    0.00|    1.15|-1234.50|
+--------+--------+--------+--------+
|  -44.00|    0.50|   -0.50|    0.10|
+--------+--------+--------+--------+
   (4$<'0.2') fmt y    NB. вычисленное w, 2 знака после точки, для каждого столбц
+------+--------+------+--------+
|  1.23|12345.00|123.40|    0.12|
+------+--------+------+--------+
|    __|    0.00|  1.15|-1234.50|
+------+--------+------+--------+
|-44.00|    0.50| -0.50|    0.10|
+------+--------+------+--------+

   NB. с использованием различных модификаторов
   (4$<'m<(>n<)>q< >0.2') fmt y
+-------+---------+-------+---------+
|  1.23 |12345.00 |123.40 |    0.12 |
+-------+---------+-------+---------+
|     __|    0.00 |  1.15 |(1234.50)|
+-------+---------+-------+---------+
|(44.00)|    0.50 | (0.50)|    0.10 |
+-------+---------+-------+---------+
   (4$<'cs<, .,>b<nil>d<n/a>0.2') fmt y
+------+---------+------+---------+
|  1,23|12 345,00|123,40|     0,12|
```

```
+------+---------+------+---------+
|   n/a|      nil|  1,15|-1 234,50|
+------+---------+------+---------+
|-44,00|     0,50| -0,50|     0,10|
+------+---------+------+---------+

   NB. fmt1 и fmt2
   'b<nil>d<n/a>0.2' fmt1 y
+--------+--------+--------+--------+
|    1.23|12345.00|  123.40|    0.12|
|     n/a|     nil|    1.15|-1234.50|
|  -44.00|    0.50|   -0.50|    0.10|
+--------+--------+--------+--------+
   'b<nil>d<n/a>c11.2' fmt2 y
       1.23  12,345.00     123.40       0.12
        n/a         nil       1.15  -1,234.50
     -44.00        0.50      -0.50       0.10
```

Глобальные параметры

9!:

9!:(2*n) запрашивает значение параметра, а 9!:(1+2*n) (если определен) устанавливает его. В версиях J с графическим интерфейсом значения этих параметров можно изменить при помощи меню Edit|Configure...

9!:0 **Затравка ГСЧ**. См. Бросить/Сдать (?).
y
9!:1
y

9!:2 **Представление по Умолчанию**. Представление(я),
y используемые по умолчанию для отображения
9!:3 не-существительных. Сами представления определены в 5!:n : 1
y атомное, 2 коробочное, 4 древовидное, 5 линейное, 6 скобочное.

9!:6 **Символы для Отображения Упаковок**. Одиннадцать
y символов, используемые при отображении упаковок (по
9!:7 умолчанию +++++++++|-).
y

9!:8 **Сообщения об Ошибках**. позволяет заменить, например,
y английские сообщения(по умолчанию) французскими.
9!:9
y

9!:10 **Точность при Выводе**. Точность, используемая при печати
y чисел (по умолчанию 6). В некоторых случаях точность можно
9!:11 установить настройкой, как в: ":!.p
y

9!:12 **Тип Системы**.
y
- 5 Unix
- 6 Windows32 (95/98/2000/NT)
- 7 Windows CE
- _1 Другая

9!:14 **Версия J**. Например:

```
y       9!:14 ''
      4.01/1998-03-15/10:24
```

9!:16 **Выравнивание Коробок (Упаковок)**. y имеет вид `r,c` задавая
y выравнивание строк и столбцов: 0 (вверх, налево); 1
9!:17 (центрировать); 2 (вниз, направо)
y

9!:18 **Точность Сравнения**. Запрашивает и устанавливает
y погрешность, используемую при сравнении чисел. См. Равно (=).
9!:19 В некоторых случаях погрешность можно изменить настройкой,
y как в: `=!.t` .

9!:20 **Предельный Размер**. Верхний предел (в байтах) для обьема
y памяти, выделяемого в каждом отдельно взятом запросе (на
9!:21 выделение памяти). По умолчанию предел равен 2^30 на
y 32-битных архитектурах и 2^62 на 64-битных архитектурах.

9!:24 **Уровень Защиты**. Уровень защиты имеет значение 0 или 1. По
y умолчанию 0, но может быть установлен в 1 (без возможности
9!:25 сбросить его обратно в 0). Когда уровень защиты равен 1,
y выполнение некоторых команд Оконного Драйвера и некоторых
внешних союзов (`!:`) , способных изменить внешнее состояние,
вызывает ошибку "security violation". Запрещены следующие
глаголы (порождаемые *внешним* союзом): диады `0!:n`, `1!:n`
кроме `1!:40`, `1!:41`, и `1!:42` , `2!:n` , и `16!:n` .

9!:26 **Фраза для Непосредственного Исполнения**. См. `9!:28` и
y `9!:29` .
9!:27
y

9!:28 **Флаг Непосредственного Исполнения**. Если этот флаг
y установлен в 1, тогда при входе в режим непосредственного
9!:29 исполнения он устанавливается в 0 и выполняется фраза для
y непосредственного исполнения (`9!:27`) .

9!:32 **Предел на Время Выполнения**. Представляет из себя
y неотрицательное скалярное (возможно не целое) количество
9!:33 секунд. Установленный таким образом предел уменьшается при
y выполнении каждой фразы в непосредственном режиме, если
это время превышает разрешающую способность таймера. Если
установленный изначально ненулевой предел достигает нуля,
выполнение прерывается с ошибкой "time limit". Текущие
ограничения:

- разрешающая способность таймера порядка миллисекунд.
- В Windows предел ограничивает полное время, использованное на работу J, а не чистое процессорное время, как изначально задумывалось.
- В Windows разрешающая способность порядка 1.5 секунд.

Например:

```
   9!:32 ''                    запросить предел; он не установлен
0
   9!:33 ]5.25                 установить предел в 5.25 секунд
   9!:32 ''                    текущее значение
5.25

   # %. ? 100 100 $ 1e6        ниже разрешающей способности
100
   9!:32 ''                    предел не изменился
5.25

   ".10#,:'# %. ?(2#100)$1e6'  разрешения достаточно
100 100 100 100 100 100 100 100 100 100
   9!:32 ''                    предел уменьшился
2.226

   ".10#,:'# %. ?(2#100)$1e6'  предел превзойден
|time limit
|   #    %.?100 100$1000000
   9!:32 ''
0
```

9!:34 **Предположения**. 1 если (и только если) предположения
y проверяются, по умолчанию 1. См. ключевое слово assert. .
9!:35
y

9!:36 **Управление Выводом**. 4-х элементный вектор, управляющий
y выводом в интерактивной сессии:

9!:37

символы конца строки	0 перевод строки (LF); 2 возврат каретки, перевод строки (CRLF)
максимальная длина строки	Строки вывода обрезаются на этой длине с добавлением "...".
макс. число строк в начале	Если полное количество строк при печати превосходит сумму чисел "макс. число строк в начале" b и "макс. число строк в конце" a , тогда выводятся первые b строк, строка "...", и последние a строк.
макс. число строк в конце	См. выше.

По умолчанию параметры вывода установлены в `0 256 0 222` .

9!:38 y 9!:39 y **Размер Хеш-Таблицы Полей Имен**. 2-х элементный вектор, управляющий начальным размером хеш-таблиц в именованых и нумерованых полях имен. По умолчанию `3 2` . Размер `i` означает, что таблица будет иметь размер `w*2^6+i` байт, где `w` -- число байт в машинном слове. Большая хеш-таблица увеличивает производительность; вне зависимости от размера хеш таблицы, поле имен может содержать по-сути неограниченное количество имен.

В следующей таблице приведены начальные значения размеров хеш-таблиц:

Категория	Индекс	# Элементов
именованые поля имен	3	499
нумерованые поля имен	2	241
таблица всех именованых полей	3	499
поля имен явных определений	1	113
`базовое поле имен`	5	2029
`z`	7	8179

Резмер хеш-таблицы поля имен может быть задан при его создании, если оно осуществляется диадной формой глагола `18!:3` .

9!:40 y 9!:41 y **Сохранять Комментарии и Пробелы**. Указывает -- сохраняются ли комментарии и несущественные пробелы в явных определениях. По умолчанию 1 (*сохранять* комментарии и пробелы). Значение 1 может привести к тому, что явные определения (даже не содержащие комментариев) потребуют для своего сохранения двойной обьем памяти.

9!:42 y **Выбор ГСЧ.** См. Бросить/Сдать (?).

9!:43 y

9!:44 y **Состояние ГСЧ.** См. Бросить/Сдать (?).

9!:45 y

9!:46 y **Файл "Остановки".** Имя файла, используемого для обработки "остановок".

```
9!:47
y
```

```
9!:48
y
9!:49
y
```

Имена Явных Аргументов. Это временная функция, позволяющая установить -- допустимы ли имена `x. y. u. v. m. n.` в явных определениях.

11!:0 y **Драйвер Окон**. См. отдельное руководство.

Отладка 13!:

См. Раздел II.J и сценарий system\main\debug.ijs.

13!:0 y **Сброс**. Сбросить стек и выключить (0) или включить (1) прерывание. Практически все подсистемы семейства 13!: требуют, чтобы прерывание было включено; все примеры ниже предполагают, что прерывание включено: 13!:0]1 . Только именованные определения (глаголы, наречия, или союзы) могут быть прерваны.

13!:1 y **Показать Стек**. В стек помещаются только именованые определения (глаголы, наречия, или союзы). См. также 13!:13 и 13!:18 .

13!:2 y **Запросить Ловушки**.

13!:3 y **Установить Ловушки**. Ловушки устанавливаются по имени и номеру строки в аргументе y, содержащем ноль или более описаний ловушек, разделенных точками с запятой. Каждое описание указывает имя, номера строк (если есть) в монадном случае, двоеточие, и номера строк (если есть) в диадном случае. Звездочка обозначает "все", и тильда обозначает "кроме". Например:

13!:3 'f 0'	f строка 0 монадного случая
13!:3 'f :2'	f строка 2 диадного случая
13!:3 'f 0 2:1'	f монадный 0 2, диадный 1
13!:3 'f 0; g :*'	f монадный 0 и g все строки диадного
13!:3 '* 0:0'	монадный 0 и диадный 0
13!:3 'a* *:*; ~ab* *:*'	Все монадные и диадные случаи определений, имена которых начинаются на a, кроме тех, имена которых начинаются на ab

```
   f=: 3 : 0
10
11
:
20
)
   13!:3 'f 1:0'      Прервать f на стр. 1 монады, и стр. 0 диады
   f ''
|stop: f
|       11
```

```
|f[1]
      13!:0 ]1        Очистить стек и включить прерывание
   3 f 4
|stop: f
|       20
|f[:0]
```

13!:4 y **Продолжить Снова**. Продолжить выполнение с текущей строки. Например:

```
   g=: 3 : ('t=. 2*y'; '1+t')
   3 4,g 'abc'
|domain error: g
|   t=.2    *y
|g[0]
      y              отступ в 6 пробелов означает прерывание
abc                  Локальное значение y
      y=. 25         Переопределить локальное значение y.
      13!:4 ''       Продолжить, попытавшись выполнить снова
3 4 51
```

13!:5 y **Продолжить Дальше**. Продолжить выполнение со следующей сроки. Например:

```
   h=: 3 : ('t=. 2 3*y'; '1+t')
   3 4,h 5 6 7
|length error: h
|   t=.2 3    *y
|h[0]
      t=. 99         отступ в 6 пробелов означает прерывание
      13!:5 ''       Продолжить дальше
3 4 100
```

13!:6 y **Выйти и Вернуть**. Выходит из глагола/наречия/союза на вершине стека, возвращая результат y . Например:

```
   g=: 3 : ('t=. 2*y'; '1+t')
   3 4,g 'abc'
|domain error: g
|   t=.2    *y
|g[0]
      13!:6 [9       Выйти из g с результатом 9
3 4 9
   h=: 2&*
   3 4,h 'abc'
|domain error: h
|h[0]
      13!:6 [97      Выйти из h с результатом 97
3 4 97
```

13!:7 y **Продолжить На**. Продолжить выполнение со строки под номером y

13!:8 [x] y **Выдать ошибку**. Выдать ошибку с кодом {.,y (целое от 1 до 255) и (необязательным) текстовым сообщением x

[x] 13!:9 y

Выполнить Еще Раз. Продолжить выполнение, выполнив еще раз неявный глагол на вершине стека с указанными аргументами. Таким образом:

```
   plus=: +
   plus/'abc'
|domain error: plus
|plus[:0] *
      13!:13 ''        Интерпретацию стека см. ниже
+----+-+-+-+-++-----+--+-+
|plus|3|0|3|+||+-+-+|  |*|
|    | | | | |||b|c||  | |
|    | | | | ||+-+-+|  | |
+----+-+-+-+-++-----+--+-+
      2 (13!:9) 3      Еще раз, получая другую ошибку
|domain error: plus
|plus[:0] !
      13!:13 ''        Обратите внимание на аргументы ('a' и 5)
+----+-+-+-+-++-----+--+-+
|plus|3|0|3|+||+-+-+|  |*|
|    | | | | |||a|5||  | |
|    | | | | ||+-+-+|  | |
+----+-+-+-+-++-----+--+-+
      1 (13!:9) 5      Еще раз
6
```

13!:11 y

Код Ошибки. Код последней ошибки. Для работы не требует включенного прерывания.

13!:12 y

Сообщение об Ошибке. Сообщение о последней ошибке. Для работы не требует включенного прерывания.

13!:13 y

Стек. Производит матрицу из 9-ти столбцов с информацией о (именованых) глаголах/наречиях/союзах в стеке:

0. Имя
1. Код ошибки или 0 если ее не было
2. Номер строки
3. Класс имени: 3, 1, или 2, соотв. глаголу, наречию или союзу
4. Линейное представление обьекта
5. Имя определяющего сценария
6. Аргумент(ы) упакованные индивидуально
7. Локальные имена в виде матрицы из 2-х столбцов: имени и значения
8. * если начало прерывания; иначе пробел

В столбцах 6 и 7, существительные приводятся "как есть", а глаголы, наречия и союзы в их линейном представлении.

Например:

```
   mean=: sum % #
   sum=: plus/
   plus=: 4 : 'x+y'
   mean 'abcd'
|domain error: plus
|   x    +y
|plus[:0]
```

```
      13!:13 ''
```

Заметьте, неявные о. не имеют локальных

```
+----+-+-+-+---------++------+-----+-+
|plus|3|0|3|4 : 'x+y'||+-+-+ |+-+-+|*|
|    | | | |         |||c|d| ||x|c|| |
|    | | | |         ||+-+-+ |+-+-+| |
|    | | | |         ||       ||y|d|| |
|    | | | |         ||       |+-+-+| |
+----+-+-+-+---------++------+-----+-+
|sum |0|0|3|plus/    ||+----+|     | |
|    | | | |         |||abcd||     | |
|    | | | |         ||+----+|     | |
+----+-+-+-+---------++------+-----+-+
|mean|0|0|3|sum % #  ||+----+|     | |
|    | | | |         |||abcd||     | |
|    | | | |         ||+----+|     | |
+----+-+-+-+---------++------+-----+-+
```

13!:14 y — **Запросить Латентное Выражение**.

13!:15 y — **Установить Латентное Выражение**. Латентное выражение выполняется перед тем как выполнение вот-вот прервется; сообщения об ошибках не выводятся; any continuation must be programmed in the latent expression.

13!:17 y — **Запросить**. Включено ли прерывание ? (Устанавливается 13!:0)

13!:18 y — **Стек как Текст**. Как 13!:1, но выдает стек в виде текстовой таблицы.

Прерванное состояние есть режим непосредственного исполнения с непустыым стеком исполнения. Новое прерванное состояние создается, когда именованный обьект (глагол, наречие, или союз) выполняется внутри прерванного состояния, и, в свою очередь, прерывается.

Доступны еще четыре глагола. Они являются частью отладочной подсистемы интегрированной среды языка J и обычно не вызываются явно. Их поведение и доступность могут быть изменены в будущем без

предупреждения.

13!:19 у — **Отрезать Назад.** Отрезать верхний уровень стека, останавливаясь на первой строке, исполняемой на следующем уровне.

[x] 13!:20 у — **Перешагнуть.** Выполнить текущую строку (или строку x, если указана) до завершения, останавливаясь на следующей строке.

[x] 13!:21 у — **Войти.** Выполнить текущую строку (или строку x, если указана), останавливаясь на следующей исполняемой строке.

[x] 13!:22 у — **Выйти.** Выполнить текущий обьект до конца, начиная с текущей строки (или строки x, если указана), останавливаясь на следующей исполняемой строке.

Динамические Библиотеки

15!:

См. главу "Динамические Библиотеки и Управление Памятью" в Руководстве Пользователя J, а так же сценарий `system\main\dll.ijs` .

x	15!:0 y	**Вызвать Функцию в Динамической Библиотеке**
	15!:1 y	**Прочитать Память**
x	15!:2 y	**Записать в Память**
	15!:3 y	**Выделить Память**
	15!:4 y	**Освободить Память**

Пространства Имен 18!:

См. также Раздел II.I и лабораторную "Locales" в меню Studio|Labs...|Locales.

18!:0 y — **Класс Названия**. Возвращает класс пространства имен с названием y, где 0 обозначает именованое, 1 нумерованое, _1 несуществующее, и _2, если слово не может служить названием пространства имен. Таким образом:

```
   18!:0 ;:'base j z 45bad asdf 0'
0 0 0 _2 _1 1
```

[x] 18!:1 y — **Список Названий**. Возвращает названия именованых (0) нумерованых (1) полей имен. Необязательный левый аргумент позволяет указать начальные буквы возвращаемых названий. Таким образом:

```
   18!:1 [0          Все именованые поля имен
+----+-+----+--------+-------+-+
|base|j|jcfg|jnewuser|newuser|z|
+----+-+----+--------+-------+-+

   asdf_bb_=: 'sesquipedalian'

   'jb' 18!:1 [0     Все именованые, с названиями на j или b
+----+--+-+----+--------+
|base|bb|j|jcfg|jnewuser|
+----+--+-+----+--------+

   18!:3 ''          Создать нумерованое поле имен
+-+
|0|
+-+

   18!:1 i.2         Все именованые и нумерованые поля имен
+-+----+-+----+--------+-------+-+
|0|base|j|jcfg|jnewuser|newuser|z|
+-+----+-+----+--------+-------+-+
```

[x] 18!:2 y — **Путь**. Монада возвращает путь поиска имен для поля с названием y ; диада устанавливает путь для поля y в x . Изначально путь имеет значение ,<,'z' , кроме пути для поля z, который изначально пуст. Если имя, искомое в поле f, там отсутствует, тогда оно ищется в полях пути f (не включая *их* пути). Например:

```
   (;:'a cd b') 18!:2 <'f'
   18!:2 <'f'
```

```
+-+--+-+
|a|cd|b|
+-+--+-+
```

Путь поля `f` установлен в `a` , `cd` , и `b` .

`[x]` `18!:3 y`

Создать. Если `y` -- пустая строка, создается нумерованое поле имен с ранее неиспользованным номером. Если `y` -- строка, тогда (вос-)создается именованое поле; если названное поле уже существует и непустое, выводится ошибка. Результатом является название созданного поля.

`x` указывает размер хеш-таблицы для поля имен, занимающей `w*2^6+x` байт, где `w` есть число байт в машинном слове. Если `x` опущено, используются умолчания, установленные `9!:38` и `9!:39` . Больший размер хеш-таблицы улучшает производительность; поле может содержать (практически) не ограниченное количество имен, вне зависимости от размера хеш-таблицы.

```
   18!:3 ''        Создать нумерованое поле
+-+
|0|
+-+
   18!:3 ''        Создать еще одно
+-+
|1|
+-+
   18!:1 [1        Названия нумерованых полей
+-+-+
|0|1|
+-+-+
```

`18!:4 y`

Переключить Текущее. По окончании выполняемого в данный момент глагола, текущим полем имен становится `y` . Изначально текущим полем является `base` .

`18!:5 y`

Текущее. Имя текущего поля. Например:

```
   18!:5 ''
+----+
|base|
+----+
```

`18!:55 y`

Стереть. По окончании выполнения глагола, стирает поле `y`. Стертое нумерованое поле нельзя использовать повторно; повторное использование именованого поля возможно без ограничений.

128!:0 у

QR. Производит разложение комплексной матрицы у (которая дол
быть инвертируема глаголом %.), на эрмитову матрицу и (квадрат
правую треугольную (QR decomposition), упакованные индивидуал

```
   mp=: +/ . *                    NB. Произведение матриц
   A=: j./?. 2 7 4$10             NB. Случ. комплексная матрица
   $A
7 4
   'Q R'=: 128!:0 A
   $Q
7 4
   $R
4 4
   >./|,(=i.4) - (+|:Q) mp Q    NB. Q эрмитова
2.70972e_16
   0~:R                           NB. R правая треугольная
1 1 1 1
0 1 1 1
0 0 1 1
0 0 0 1
   A -: Q mp R
1
```

128!:1 у

R Inv. Обращает (квадратную) правую треугольную матрицу.

x 128!:2 у

Применить. x 128!:2 у применяет глагол в строке x к у . Напри

```
   '+/' 128!:2 i.2 5
5 7 9 11 13
   '+/' 128!:2"1 i.2 5
10 35
   '+/"1' 128!:2 i.2 5
10 35
   ('+/';'|.';'|."1') 128!:2&.><i.2 5
+-----------+---------+---------+
|5 7 9 11 13|5 6 7 8 9|4 3 2 1 0|
|           |0 1 2 3 4|9 8 7 6 5|
+-----------+---------+---------+

   '2 3' 128!:2 i.2 5
|syntax error
|   '2 3'    128!:2 i.2 5

   '@' 128!:2 i.2 5
|syntax error
|   '@'    128!:2 i.2 5
```

128!:2 имеет ранги 1 _ , тоесть применяет списки в левом аргуме
к правому аргументу *полностью*.

[x] 128!:3 y

Контрольная Сумма (CRC). Многочлен CRC определяется булевским списком или целым. Следующие многочлены CRC -- p и эквивалентны:

```
   p=: 1 1 1 0  1 1 0 1  1 0 1 1  1 0 0 0  1 0 0 0  0 0 1 1  0 0 1 0  0
   q=: _306674912

   _8]\p
1 1 1 0 1 1 0 1
1 0 1 1 1 0 0 0
1 0 0 0 0 0 1 1
0 0 1 0 0 0 0 0

   q -: (_2 _2,30$2) #. p
1
   p -: (32$2) #: q
1
```

x 128!:3 y вычисляет CRC строки y в соответствии с x , который м иметь вид p или p;i , где p -- многочлен CRC, а i -- начальное знач CRC. Если i опущено, в качестве начального значения использует 128!:3 y эквивалентно _306674912 (128!:3) y .

Монада x&(128!:3) поддержана специальным кодом, который пред-вычисляет значения CRC для всех возможных значений байт использует эту таблицу для быстрого вычисления CRC строк.

Примеры:

```
   f=: 128!:3
   f '123456789'
_873187034
   f 'assiduously avoid any and all asinine alliterations'
1439575093
```

128!:4 y

Прямой Доступ к ГСЧ. См. Бросить/Сдать (?).

128!:5 y

Is NaN. Если z=: 128!:5 y , тогда z имеет такую же размерность и атом z равен 1 тогда и только тогда, когда соответствующий ато есть _. (NaN) или содержит _. .

Примеры:

```
   (128!:5) 3.45 6 _. 7
0 0 1 0
   (128!:5) t=. 2 2$1 2;3 4 5;(<<_. 7);_.
0 0
1 1
   (128!:5) < t
1
   (128!:5) ;:'Cogito, ergo sum. _.'
0 0 0 0 0
```

+ +

Приложение B. Специализированный Код

Многие примитивы содержат специализированный код для некоторых видов их аргументов, позволяющий достичь большей эффективности (по времени выполнения или обьему необходимой для этого памяти), недоступной в общем случае. Более того, некоторые "узнаваемые" фразы тоже реализованы специализированным кодом. Например, диадный случай крючка `($,)` точно соответствует оператору *reshape* в APL (обозначаемому там *ρ*); его специализированная реализация избегает разборки правого аргумента, экономя, таким образом, процессорное время и память:

```
   ts=: 6!:2 , 7!:2@]
   x=: 11 13 17 19 23
   y=: 29 7 23 11 19$'sesquipedalian'

   (x ($,) y) -: x $, y
1

   ts 'x ($,) y'
0.00773981 2.09818e6
   ts 'x $, y'
0.0170125 3.14662e6
```

Реализованные специализированным кодом распознаваемые случаи перечислены ниже:

`=`	диада	на булевских аргументах операция выполняется пословно (параллельно в рамках процессорного слова), как и для следующих глаголов: `= < <. <: > >. >: +. +: * *. *: ^ ~: | !`
`<.@f`	оба	как и `>.@f` избегает нецелых промежуточных результатов на целых произв. точности
`<.@%`	диада	так же `<.@:% >.@% >.@:%` ; спец. код для целых аргументов см. информацию об изменениях в J 6.02
`i.<./`	монада	так же `i.>./` and `i:<./` and `i:>./` ; спец. код для целых и действительных списков; см. информацию об изменениях в J 5.04
`i.>./`	монада	так же `i.<./` , `i:<./` и `i:>./` ; спец. код для целых и действительных списков (см.

		информацию об изменениях в J 5.04)
+	диада	так же * и - ; под Windows, реализована на ассемблере для целых аргументов и операций вектор-вектор, вектор-скаляр, и скаляр-вектор
^	диада	x^y вычисляется повторным умножением для действительных x и целых y
m&|@^	диада	избегает возведения в степень для целых аргументов
m&|@(n&^)	монада	избегает возведения в степень для целых аргументов
{&x^:a:	монадad	спец. код для целых списков x ; см. информацию об изменениях в J 5.04
{~^:a:	диада	спец. код; см информацию об изменениях в J 5.04
f@]^:g	диада	применяет f вместо x&(f@]) на каждой итерации; см. информацию об изменениях в J 6.01
|	диада	спец. код, когда левый аргумент -- положительная степень двойки; см. информацию об изменениях в J 6.02
+/ .*	диада	так же +./ .* и ~:/ .*. ; спец. код в общем случае; спец. код для булевского левого или правого аргумента и векторного правого аргумента; см. информацию об изменениях в J 5.04 и информацию об изменениях в J 6.01
-/ .*	монада	спец. код в общем случае; спец. код для квадратных матриц; спец. код для массивов матриц 2 на 2; см информацию об изменениях в J 4.05
$,	диада	так же ($,)"r ; так же {, {., }., e., ; избегает разборки; см. информацию об изменениях в J 4.06
x=.x,y	-	так же =: ; на месте (без перемещения данных в памяти); см. информацию об изменениях в J 5.03
;@:(<@u;.n)	диада	спец. код; см. информацию об изменениях в J 5.03
;@:{	диада	спец. код; см. информацию об изменениях в J 5.03
[;.0	оба	так же] ; спец. код для векторных и матричных правых аргументов; см. информацию об изменениях в J 5.01

`f;.1`	оба	так же `f;._1 f;.2 f;._2` ; избегает построения ячеек аргумента для некоторых глаголов: `< $ , # [ ] {. {: <@}. <@}:` ; так же `<&}. <@:}.` и т.д.; см. информацию об изменениях в J 4.05
`f;.1`	оба	так же `f;._1 f;.2 f;._2` ; спец. код для прореженных булевских левых аргументов; см. информацию об изменениях в J 4.06
`f;.1`	оба	так же `f;._1 f;.2 f;._2` ; спец. код, когда результат `f` в каждом разрезе -- упаковка или атом; см. информацию об изменениях в J 5.03
`f/;.1`	оба	так же `f;._1 f;.2 f;._2` ; спец. код для атомных глаголов `= < + +. *` и т.д.; см. информацию об изменениях в J 5.03
`f;.3`	оба	так же `f;._3` ; спец. код для матричных аргументов
`;@:(<@u;.n)`	диада	спец. код; см. информацию об изменениях в J 5.03
`#`	диада	спец. код для булевских левых аргументов
`#:`	диада	спец. код для `(0,d)#:y` , где `d` является положительной степенью двойки и `y` целое; см. информацию об изменениях в J 6.02
`#: i.@(*/)`	монада	так же `(#: i.&(*/))` , и т.д.; спец. код для неотрицательных целых векторов; см. информацию об изменениях в J 4.05
`x&(128!:3)`	монада	спец. код, пред-вычисляющий таблицу значений CRC для каждого байта
`=/"r`	монада	так же `< <: > >: +. +: * *. *: ~:` ; работает пословно для булевских аргументов; так же спец. код для вставок вдоль измерений длины 2 см. информацию об изменениях в J 5.03
`+/`	монада	так же `*` и `-` ; под Windows реализовано на ассемблере для целых аргументов
`,/`	монада	так же `,. ; ,&.> ,.&.>` ; линейное время; см. информацию об изменениях в J 4.05 и информацию об изменениях в J 7.01
`m b./`	оба	спец. код для побитных булевских функций; см. информацию об изменениях в J 5.01

f/@,	монада	так же f/@:, f/&, f/&:, ; избегает разборки; см. информацию об изменениях в J 4.05
f/@:g	диада	также [: f/ g ; специализированный код для атомных f и g ; дополнительная оптимизация для +/@:g ; см. информацию об изменениях в J 6.01
#/.	диада	избегает построения ячеек аргумента; см. информацию об изменениях в J 5.03и информацию об изменениях в J 6.02
+//.	монада	так же +. *. = ~: <. <: >. >: 17 b. 22 b. 23 b. ; избегает построения ячеек аргумента; см. информацию об изменениях в J 6.01
+//.	диада	так же +. *. <. >. = ~: n b. для различных векторных правых аргументов; избегает построение ячеек аргумента; см. информацию об изменениях в J 5.03
</. i.@#	монада	так же x</.i.#x ; спец. код; см. информацию об изменениях в J 5.03
({.,#)/.	диада	так же (#,{.)/. ; спец. код; см. информацию об изменениях в J 5.04
(+/%#)/.	диада	избегает построения ячеек аргумента; см. информацию об изменениях в J 6.02
+//.@(*/)	диада	так же ~://.@(*./) ~://.@(+./) +//.@(*./) +//.@(+./) (22 b.)//.@(17 b./) ; избегает построения ячеек аргумента; см. информацию об изменениях в J 6.01
/:	оба	так же \: ; спец. код для нескольких типов данных; спец. код для аргументов с 5-ю или менее элементами; см. информацию об изменениях в J 4.05
/:	диада	спец. код для случая, когда левый и правый аргументы -- идентичные булевские, текстовые, целые или действительные вектора; так же для /:"1 , когда левый и правый аргументы -- идентичные булевские, текстовые, целые или действительные массивы; то же для \: ; см. информацию об изменениях в J 5.01
/:~	монада	спец. код для булевских, текстовых, целых и действительных векторов; так же для /:~"1 и /:"1~ ; то же для \: ; см.

		информацию об изменениях в J 5.01
`/:@/:`	монада	так же `/:@:/:` и `/:&/:` и т.д.; спец. код; см. информацию об изменениях в J 5.04
`{/:`	диада	так же `{/:~` ; спец. код для скалярных целых левых аргументов и целых или действительных векторных правых аргументов; см.информацию об изменениях в J 6.03
`#\`	монада	спец. код; см. информацию об изменениях в J 7.01
`=/\`	монада	так же `+. *. ~:` ; работает пословно для булевских аргументов
`+/\`	монада	так же * и - ; под Windows, реализовано на ассемблере для целых аргументов
`+/\`	диада	так же `<. >. +. *. = ~: m b.` для атомных `m e. 17 22 23 25` ; спец. код для положительных левых аргументов и булевских, целых или действительных правых аргументов; см. информацию об изменениях в J 5.03 и J 6.02
`[\`	диада	так же `]` и `,` ; см. информацию об изменениях в J 5.01
`2 f/\y`	диада	спец. код; см. информацию об изменениях в J 4.06 и J 6.01
`m b./\`	монада	спец. код для побитных булевских функций; см. информацию об изменениях в J 5.01
`(+/%#)\`	диада	спец. код для положительных левых аргументов с целыми и действительными правыми аргументами; см. информацию об изменениях в J 5.03
`#\.`	монада	спец. код; см. информацию об изменениях в J 7.01
`u/\.`	монада	спец. код
`=/\.`	монада	так-же `< <: > >: +. +: *. *: ~:` ; работает пословно для булевских аргументов
`+/\.`	монада	так же * и - ; под Windows, реализовано на ассемблере для целых аргументов
`m b./\.`	монада	спец. код для побитовых булевских функций; см. информацию об изменениях в J 5.01
`f/\.`	диада	спец. код для поатомно ассоциативных глаголов `f` ; см. информацию об

		изменениях в J 6.02
{	диада	спец. код для правых аргументов некоторых типов данных; спец. код для целых левых аргументов; спец. код для индексирования первых двух измерений
<"1@[{]	диада	избегает <"1 если левый аргумент -- массив целых
p{~q i.]	монада	так же (q i.]) { p"_ и q&i.{p"_ ; спец. код, когда p и q текстовые списки; см. информацию об изменениях в J 6.02
{/:	диада	так же {/:~ ; спец. код для скалярных целых левых аргументов и целых или действительных векторных правых аргументов; см. информацию об изменениях в J 6.03
a=: c}x,y,:z	-	избегает присоединения и наклеивания; на месте, если c имеет булевский тип, и a есть x или y ; см. информацию об изменениях в J 4.05
y=: x i}y	-	на месте
y=: y i}~ x	-	не выделяет памяти; см. информацию об изменениях в J 6.03
f"r	оба	многие глаголы имеют встроенную поддержку ранга монады и диады: <. <: < >. >: + +. +: * *. *: - -: % ^ ~: | |. |: , ,: ! /: \: [ ] {. }. o. только монады: -. %: ^. #. ,. / \ \. {: }: @. ? A. H. j. p. p: q: r. t. x: /:@/: только диады: = > $ # { e. i. i: ($,) m b. -.@-: Для (+/%#)"r см. информацию об изменениях в J 6.02.
^@o.	монада	спец. код; см. информацию об изменениях в J 7.01
"."0@":	монада	так же @: или & или &: вместо @ ; специализированный код; см. информацию об изменениях в J 6.01
f0`...`fn@.v"0	монада	спец. код, если f0 , ... , fn атомные; см. информацию об изменениях в J 5.02
x&(128!:3)	монада	спец. код, предвычисляющий таблицу значения CRC для каждого байта
m&i.	монада	

		так же `m&i: -.&m e.&m` ; см. информацию об изменениях в J 5.04
`u&.(a.&i.)`	монада	спец. код если `u-:u"0` ; см. информацию об изменениях в J 6.02
`m b./&.(a.&i.)`	монада	избегает преобразования из/в целые; см. информацию об изменениях в J 6.02
`m b.&.(a.&i.)`	диада	избегает преобразования из/в целые; см. информацию об изменениях в J 6.02
`?`	монада	так же `?.` ; спец код, если аргумент равен 2
`?@$`	диада	так же `?.` ; так же `?@#` ; так же `[: ? $` ; спец. код; см. информацию об изменениях в J 5.03
`e.i.0:`	диада	см. строку для `f i.0:` ниже
`E.`	монада	спец. код для булевских и текстовых векторов
`E.i.1:`	диада	см. строку для `f i.0:` ниже
`i.`	монада	так же `i:` ; спец. код для аргументов длины 1
`i.`	диада	так же `e.` и `i:` ; спец. код для нескольких типов данных; спец. код для `i.!.0` ; спец. код для монады `i.~` или `x i. x` ; спец. код для аргументов с большим числом одинаковых столбцов; см. информацию об изменениях в J 4.05
`f i.0:`	диада	спец. код для следующих функций; см. информацию об изменениях в J 5.01, J 5.03, и J 5.04.

`f i. 0:`	`i.&0@:f`	первая позиция, где не `x f y`
`f i. 1:`	`i.&1@:f`	первая позиция, где `x f y`
`f i: 0:`	`i:&0@:f`	последняя позиция, где не `x f y`
`f i: 1:`	`i:&1@:f`	последняя позиция, где `x f y`
`[: +/ f`	`+ /@:f`	количество позиций, где `x f y`
`[: +./ f`	`+./@:f`	`x f y` где либо ?
`[: *./ f`	`*./@:f`	`x f y` везде?
`[: I. f`	`I. @:f`	позиции, где `x f y`

		Примечания: • f один из глаголов = ~: < <: > >: E. e. • Спец. код используется только для булевских, целых, действительных, текстовых и символьных аргументов • Спец. код используется только для атомов и списков, если f не e. • Если f есть E. , то спец. код только для E.i.1: +/@:E. +./@:E. I.@:E.
i.<./	монада	так же i.>./ , i:<./ и i:>./ ; спец. код для целых и действительных списков; см. информацию об изменениях в J 5.04
m&i.	монада	так же m&i: -.&m e.&m ; см. информацию об изменениях в J 5.04
p {~ q i.]	монада	так же (q i.]) { p"_ и q&i. { p"_ ; спец. код, когда p и q текстовые списки; см. информацию об изменениях в J 6.02
I.@:<	диада	см строку для f i.0: выше
+/%#	монада	спец. код для следующих случаев: (+/%#)/. J 6.02 (+/%#)\ J 5.03 (+/%#)"r J 6.02

Приложение C. Системные Соглашения и Пределы

код ошибки	(правый аргумент `13!:8`) меньше `256`
точность печати	меньше или равно `20`
погрешность	меньше или равно `2^_34`
предел рекурсии	(ограничение на вложенность вызовов) 100 для Windows CE, иначе 10000

Значимая часть массивов внутренних булевского, текстового или Unicode типов оканчивается дополнительным нулевым байтом.

Приложение D. Ошибки

Далее перечислены ошибки, которые могут быть выданы системой.

Текст сообщения	Пример	Описание
allocation error		недостаточно места в от (memory-mapped) упако
assertion failure		T не массив из всех 1 в у конструкции assert. T
attention interrupt		
break		
control error		неверно оформленная у конструкция; или резул управляющей конструкц существительным.
domain error	`3+'4'`	
file access error		нет разрешения на выпо файлом
file name error	`1!:1 <'foojunkfoo'`	названный файл не суще
file number error		
ill-formed name	`7!:5 <'bad name!'`	нарушает правила запис определено в Главе I
ill-formed number	`1234 4q3`	нарушает правила запис как определено в Главе Констант
index error	`19 { i.7`	значение индекса выход пределы
input interrupt		
interface error		
length error	`2 3 + 4 5 6`	
limit error	`i. 9e8`	
locale error		
nonce error		не реализовано
non-unique sparse elements	`($.i.4){i.4 2`	элемент прореживания массива должен быть ат
open quote	`'don't do it'`	выполненная строка сод количество кавычек вне
out of memory		не хватает память; или использования памяти,
rank error	`(i.2 3) i. 1`	
read-only data		

security violation		операция не разрешена защиты; см. 9!:24 и 9!:
spelling error	`3 z: '4'`	нарушает правила слово описаны в Разделе I) ил примитив, значение кот
stack error	`".t=:'".t'`	превышен предел рекур переопределение прерв наречия или союза (кро
stop		
syntax error	)	результат предложения существительным/глагол глагол, пытающийся пр виде глагола/наречия/с
system error		"невозможная" ошибка внутреннего контроля ц
time limit		превышен предел на вре 9!:32 и 9!:33
value error	`Uninitializedname`	использование имени, к присвоено значения