Concrete Math Companion

Kenneth E. Iverson

TABLE OF CONTENTS

Chapter 0
NOTATION

This book is written as a companion to *Concrete Mathematics* (Graham, Knuth, and Patashnik [1]); following it closely in its choice of topics and order of treatment, and making frequent explicit references to it. Because this book is written in an executable notation, any expression can be entered directly on a computer for experimentation.

This ability to experiment with the mathematical ideas complements the treatment in GKP. Although this text can be used independently, it is recommended that the texts be used together, sometimes reading a section from GKP first, and sometimes reversing the order.

Conventional mathematical notation is *analytic* in the sense that it permits meaningful manipulation of sentences according to relatively strict and simple rules. The use of notation that is both analytic and immediately executable on a computer permits quick and reliable experimentation that can make abstract mathematical ideas more accessible, and their study more enjoyable.

The notation **J** used in this text possesses these properties, and is readily available on a wide variety of computers.

In order to minimize digressions from the mathematical development, notation will be introduced together with brief commentary sufficient to interpret the particular sentence. To achieve a broader understanding of the notation a reader may:

a) Experiment by entering related sentences on the computer.

b) Consult the final section of each chapter for discussion of the notation introduced.

c) Consult ***J*** *Introduction and Dictionary* (Iverson [2]).

This style of development will be illustrated by introducing a few of the basic elements of the notation, using a fixed-width font for dialogue with the computer, and Times Roman for commentary:

```
   a=: 0 1 2 3 4
```
The copula `=:` assigns a name to any entity

```
   a % 10
0 0.1 0.2 0.3 0.4
```
Division is denoted by `%`

```
   +/ a
10
```
Adverb `/` inserts its verb argument `+` between items

```
   sum=: +/
   sum a
10
```
The verb or function `sum`

```
   # a
5
```
Tally, or number of items

```
   mean=: sum % #
   mean a
2
```
The arithmetic mean or average

```
   a +/ a
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
```
The function `+/` is *ambivalent*; applied monadically (to a single argument, as in `+/ a` above) it produces summation; applied dyadically (as it is here) it produces a function table (in this case an addition table). In this it mimics the use of the symbol `-` in math, which represents negation or subtraction as dictated by its context.

```
   a ; 2 * a
┌─────────┬─────────┐
│0 1 2 3 4│0 2 4 6 8│
```
A two-element list of *boxed* results is produced by `;`

```
└────────┴────────┘
```

```
   a (+/ ; %/ ; ^/ ; !/) a
```

Four boxed tables (_ denotes infinity)

```
┌─────────┬─────────────────────────────┬─────────────────┬─────────┐
│0 1 2 3 4│0 0   0         0    0│1 0  0  0   0│1 1 1 1 1│
│1 2 3 4 5│_ 1 0.5 0.3333333 0.25│1 1  1  1   1│0 1 2 3 4│
│2 3 4 5 6│_ 2   1 0.6666667  0.5│1 2  4  8  16│0 0 1 3 6│
│3 4 5 6 7│_ 3 1.5         1 0.75│1 3  9 27  81│0 0 0 1 4│
│4 5 6 7 8│_ 4   2   1.33333    1│1 4 16 64 256│0 0 0 0 1│
└─────────┴─────────────────────────────┴─────────────────┴─────────┘
```

```
   (i.4);(i.3 4);(i.2 3 4);(+/i.2 3 4)
┌───────┬─────────┬───────────┬───────────┐
│0 1 2 3│0 1  2  3│ 0  1  2  3│12 14 16 18│
│       │4 5  6  7│ 4  5  6  7│20 22 24 26│
│       │8 9 10 11│ 8  9 10 11│28 30 32 34│
│       │         │           │           │
│       │         │12 13 14 15│           │
│       │         │16 17 18 19│           │
│       │         │20 21 22 23│           │
└───────┴─────────┴───────────┴───────────┘
```

```
   cube=: ^&3
```

The conjunction `&` bonds *power* to a right argument

```
   cube a
0 1 8 27 64
```

The result is a monadic function

```
   trin=: 2&!
```

The conjunction `&` bonds similarly to a left argument

```
   trin a
0 0 1 3 6
```

Triangular numbers

```
   2: a
2
```

The constant function `2:` applies to entire argument

```
   2: " 0 a
2 2 2 2 2
```

The same function of rank zero applies to each atom

We conclude these samples of notation with the *tie* conjunction (`` ` ``) that applies to verbs to produce a *gerund* (a noun that carries the force of a verb), and the *agenda* (`@.`) that selects for action one of the verbs that comprise a gerund:

```
   +`* / a
14
```

```
   0+1*2+3*4
14
```

Unparenthesized sentences are executed from right to left; there is no hierarchy among functions

```
   0+(1*(2+(3*4)))
14
```

```
   !`^@.(<&0) 4
24
```

Exponential of negative arguments; factorial of others

```
   !`^@.(<&0) -4
0.0183156
```

A. THEOREMS AND PROOFS

A *theorem* is an assertion that one expression `L` (the *left limb*) is equivalent to another `R`, and may be expressed as the function `T=: L -: R` . A theorem may also be called a *tautology*, a function that yields `1` (true) for any argument. For example:

```
   L1=: +/@i.
```

Sum of integers

```
   R1=: ([ * ] - 1:) % 2:
   T1=: L1 -: R1
   (T1 ; L1 ; R1 ; i.) 6
┌─┬──┬──┬───────────┐
│1│15│15│0 1 2 3 4 5│
```

We can also assign the name `n` to the right argument function `]` to allow a function such as `R1` to be written more readably for a beginner. Thus:

```
n=: ]
R1=: (n*n-1:)%2:
```

A *proof* is a sequence of equivalent expressions that lead in justifiable steps from a left limb to a right. We will write one expression below another to assert that it is equivalent to the one above it, possibly annotating it with the justification to provide a *formal* proof:

`L1`	Theorem 1
`+/@i.`	Definition of `L1`
`+/@|.@i.`	Sum is assoc and comm (`|.` is reversal)
`((+/@i.)+(+/@|.@i.)) % 2:`	Half of sum of equal quantities
`+/@(i. + |.@i.) % 2:`	Summation distributes over addition
`+/@(n # n - 1:) % 2:`	Sum with reversal gives list of `n-1`
`(n * n - 1:) % 2:`	Definition of times
`R1`	Definition of `R1`

We will also present proofs beginning with the theorem and continuing with the sequence leading from the left limb to the right. For example:

`odds=: 1: + 2: * i.`	First odd integers
`T2=: (+/@odds) -: *:`	Theorem 2: sum of odds gives square
`+/@(1: + 2: * i.)`	Definition of `odds`
`n + +/@(2: * i.)`	Sum of `n` ones is `n`
`n + 2: * +/@i.`	Sum of twice is twice sum
`n + 2: * (n * n - 1:) % 2:`	Theorem 1
`n + n * n - 1:`	Simple algebra
`n * n`	Simple algebra
`*:`	Definition of square

We will use names such as `GKP5_4` to refer to theorem 4 of chapter 5 of GKP. Thus:

```
GKP5_4=: L5_4 -: R5_4=: (-~ ! ])"0 [. L5_4=: !
i (L5_4/ ; R5_4/ ; GKP5_4/) i=: i.6
┌────────────────┬────────────────┬─┐
│1 1 1 1 1  1│1 1 1 1 1  1│1│
│0 1 2 3 4  5│0 1 2 3 4  5│ │
│0 0 1 3 6 10│0 0 1 3 6 10│ │
│0 0 0 1 4 10│0 0 0 1 4 10│ │
│0 0 0 0 1  5│0 0 0 0 1  5│ │
│0 0 0 0 0  1│0 0 0 0 0  1│ │
└────────────────┴────────────────┴─┘
```

Chapter 1
RECURSION

If a function recurs in the expression that defines it, the function is said to be recursively defined. Such a definition must be supplemented by a definition for some specific argument, using an expression that does not make use of the function being defined.

For example, the factorial of the argument `j` may be defined by `j * f j-1` (or more formally by `] * f@<:`), supplemented by the definition `1:` for the case `j=: 0`. Thus:

```
    f=: 1:`(]*f@<:) @. *

   f 5
120

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

The function `f` is applied to each rank `0` cell of `i. 6`, that is, to each scalar

In the foregoing definition, the signum function `*` yields 0 if the argument is zero, and 1 if it is greater than zero. Consequently, the agenda `@.` chooses the last element of the gerund ``1:`(]*f@<:)`` each time until the argument (repeatedly decremented by `<:`) becomes zero, in which case it chooses the constant function `1:`, thus terminating the process.

Alternatively, the imposition of zero rank could be incorporated in the recursive definition :

```
    f=: 1:`(]*f@<:) @. * " 0

    f i. 6
1 1 2 6 24 120
```

The reference to `f` within the definition works only because the name `f` is assigned to the function defined; we may instead use the symbol `$:` for *self-reference* to define an anonymous function to which any name may be assigned:

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

   factorial=: 1:`(]*$:@<:) @. * " 0

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

A. THE TOWER OF HANOI

In this puzzle, discs are to be moved from post A to post B using post C, a larger disc never being placed on a smaller. The two expressions in GKP1.1 (Eq 1.1 of GKP) for the number of moves required for `n` discs lead to the following recursive definition:

```
    T=: 0:`(1:+2:*T@<:) @. * " 0
    T x=: i. 10
0 1 3 7 15 31 63 127 255 511
```

This result suggests experiments that lead to an equivalent non-recursive definition:

```
   1+T x
1 2 4 8 16 32 64 128 256 512
   2^x
1 2 4 8 16 32 64 128 256 512
   t=: (2: ^ ]) - 1:                    (Or t=: <:@(2&^))
   t x
0 1 3 7 15 31 63 127 255 511
   (T = t) x
1 1 1 1 1 1 1 1 1 1
   (T -: t) x                           A tautology (true for any argument)
1
```

In the definition of `t` above, the right bracket symbol `]` denotes the right argument. By assigning the name `n` to it we can also use notation that may be easier to compare with the expressions in GKP. Thus:

```
   n=: ]
   t=: (2: ^ n) - 1:
```

Properties of a function that is defined recursively can often be established inductively: a property is assumed to hold for a specific argument value `j`, and this assumption is used to prove that it must then hold for the argument `j+1`. It then remains to show that it does indeed hold for some specific value of the argument.

For example, we will assume that the functions `T` and `t` agree for the argument `j`. Stated formally:

```
   (T=t) j                         Induction hypothesis
1
```

We now establish that `(T=t) j+1` is therefore true:

```
   (T=t) j+1
   (T j+1)=(t j+1)                     Definition of the fork T=t
   (T j+1)=((2^j+1)-1)                 Definition of t
   (T j+1)=((2*2^j)-1)                 Definition of power
   (T j+1)=(1+2*t j)                   Def of t and simple algebra
    (1+2*T j)=(1+2*t j)                Definition of T
   (T j)=(t j)                         Simple algebra
   (T = t) j                           Definition of fork
   1                                   Induction hypothesis
```

If a typical value is assigned to the argument `j` that appears in the proof above (as in `j=: 10`), then each line of the proof (excluding the comments) may be entered on the computer to yield the result `1`. If each equal sign is replaced by a comma or semicolon, the result will be the results of each limb of the assertion. Such displays can be helpful in developing a proof, since a false step will probably show a discrepancy.

Any proof can be so *illuminated* by entering the steps, with the argument or arguments appended. For example, after entering `x=: 10`, the first three lines of the proof of Theorem 1 in Chapter 0 may be entered with `x` appended. Because it is a fork, the next line must first be enclosed in parentheses.

Since the functions agree at zero, they must agree for all succeeding integer arguments. The pattern shown by the result of `1+T i.10` was so obvious as to require no explicit analysis of the relations between successive elements in order to define an equivalent `t`, but we will use it to illustrate methods that will be so used. Thus:

```
   a=: 1 + T i. 10

   a
1 2 4 8 16 32 64 128 256 512

   2 +/\ a                         Sums of successive pairs
3 6 12 24 48 96 192 384 768

   3 -/\a                          Alternating sums of triples
3 6 12 24 48 96 192 384

   2 %/\ a                         Division over pairs
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5

   2 %~/\ a                        Commuted division over pairs
2 2 2 2 2 2 2 2 2
```

Recursion also provides a simple definition of the sequence of moves in Hanoi. We will first present and apply such a definition, and then use it to illustrate the steps in *reading* or *interpretation*:

```
   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
```

See §E if this definition does not work

```
    0 1 2 3 4 <@h"0 1 x
```

	A	AAC	AABACCA	AACABBAACCBCAAC
	B	CBB	BCCBABB	CBBCACCBBAABCBB

The foregoing definition uses some new notation, but the first line makes it clear that `h` is recursively defined, with a base `b` that is selected when the condition `c` produces a zero, and with a main part `(p,.q,.r)` selected when it produces a one (that is, when the left argument exceeds one). The main part uses the catenation `,.` which should be experimented with if unfamiliar:

```
   (3 4,.5 6) ; (1 2,.3 4,.5 6)
```

3 5	1 3 5
4 6	2 4 6

The inner function `q` is simply the movement of a single disc, but the outer functions employ the possibly unfamiliar permutation primitive `A.` :

```
   (0 1 2 3 4 5 A. 'ABC');(0 1 2 3 4 5 A. i.3)
```

ABC	0 1 2
ACB	0 2 1
BAC	1 0 2
BCA	1 2 0
CAB	2 0 1
CBA	2 1 0

```
   0 1 2 3 4 5 A. 'first';'second';'third'
```

first	second	third
first	third	second
second	first	third
second	third	first
third	first	second
third	second	first

The functions `p` and `r` are therefore seen to be recursive applications of the function `h` with the left argument (number of discs) decremented, and with the right argument (the posts) permuted.

The expression `g=: h f.` may be used to produce an equivalent function `g`; the adverb `f.` applies to its argument `h` to (recursively) substitute the referent of each name encountered, producing a definition in terms of primitives only.

B. TRIANGULAR NUMBERS

The function `trn=: +/@Ai` uses the function `Ai=: 1:+i.` (Augmented indices) to produce the *triangular* numbers, defined as the number of coins in a packed triangular array with a specified number of coins in the base row. For example:

```
   (Ai=: 1:+i.) 4
1 2 3 4
   trn 4
10
   trn"0 i. 15
0 1 3 6 10 15 21 28 36 45 55 66 78 91 105
```

Since `trn j` equals `j+trn j-1`, an equivalent recursive definition is:

```
   sr=: 0:`(n + $:@<:)@.*
```

The function `S=: (n*n+1:)%2:` given by GKP1.5 (or the equivalent `2:!>:`) can be shown to be equivalent to the recursive definition `sr` by an inductive proof. We offer instead a proof of the equivalence to `S`, based on the observation of Gauss cited in GKP; prefacing it with illustrations of some of the expressions to be used in the proof:

```
   S=: (n*n+1:)%2:                  Recall that n=: ]
   j=: 10
   (trn,sr,S) j
55 55 55
   |. Ai j
10 9 8 7 6 5 4 3 2 1
   (Ai + |.@Ai) j
11 11 11 11 11 11 11 11 11 11
   j # j+1
11 11 11 11 11 11 11 11 11 11
   +/ j # j + 1
110
   j * j + 1
110
   (j*j+1)%2
55
```

Proof:

```
   trn j
   +/@Ai j                          Definition of trn
   +/@|.@Ai j                       +/ is symmetric (See § E)
   -:@(+/@Ai + +/@|.@Ai) j          Half sum of equals
   -:@(+/)@(Ai + |.@Ai) j           Sum distributes over +
   -:@(+/)@(] # ] + 1:) j           Sum is a list of constants
   -:@(] * ] + 1:) j                Definition of multiplication
   ((] * ] + 1:) % 2:) j            Definition of -: (halve)
   ((n * n + 1:) % 2:) j            Definition of n
   S j                              Definition of S
```

C. THE JOSEPHUS PROBLEM

If `f=: }:@|.` then the repeated application of `i&f` to an argument `x` removes items located at intervals of `i` from those remaining (treated as a circle):

```
   f=: }:@|.                        Delete last after rotating
   x=: 'ABCDE'
   x;(3 f x);(3 f 3 f x);(3 f^: 0 1 2 3 x)
```

ABCDE	DEAB	BDE	ABCDE DEAB BDE BD

The original Josephus problem as presented in GKP concerns the application of `3&f` to the argument `Ai 41` (the positions of 41 men formed in a circle) until only two (that is, one less than the interval) survive. To effect this we define and use the following function:

```
   js =: f^:(1: + #@] - [)
   3 js Ai 41
16 31
```

We will concentrate (as does GKP) on the case of an interval of two, and therefore define a function `j` that is equivalent to `js` except that it has a monadic case `2&js` and ranks `0 1`:

```
   j=: 2&$: : js " 0 1
   (j x);(,j\x);(,j\1+i.# x);(,j\i.# x)
```

C	AACAC	1 1 3 1 3	0 0 2 0 2

```
   ,j\Ai 16
1 1 3 1 3 5 7 1 3 5 7 9 11 13 15 1
   < ;. 1 ,j\Ai 16    Box cut on leading item. See §E.
```

1	1 3	1 3 5 7	1 3 5 7 9 11 13 15	1

The last two results agree with the table of values of `j` that appears after GKP1.8. However, a more obvious pattern is provided by labelling the positions with indices beginning at zero rather than one, and we will continue to use such zero-origin indexing hereafter:

```
   < ;. 1 ,j\i. 16
```

0	0 2	0 2 4 6	0 2 4 6 8 10 12 14	0

This result leads to a recursive definition that differs somewhat from that of GKP1.8, but can easily be related to it:

```
   jr=: 0:`even`odd @. c
   even=: +:@>:@jr@<:@-:
     odd=: +:@jr@-:@<:
     c=: * * >:@(2&|)
   b ,: jr"0 b=: i. 20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 0 2 0 2 4 6 0 2 4  6  8 10 12 14  0  2  4  6  8
   <;.1 jr"0 b
```

0	0 2	0 2 4 6	0 2 4 6 8 10 12 14	0 2 4 6 8

The pattern produced by `jr` may be examined further as follows:

```
   b=: i.<:2^5
   ,. y=: <;.1 jr"0 b
┌─────────────────────────────────────────────┐
│0                                            │
├─────────────────────────────────────────────┤
│0 2                                          │
├─────────────────────────────────────────────┤
│0 2 4 6                                      │
├─────────────────────────────────────────────┤
│0 2 4 6 8 10 12 14                           │
├─────────────────────────────────────────────┤
│0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30   │
└─────────────────────────────────────────────┘
```

```
   #&>y                                   Lengths of blocks
1 2 4 8 16

   +/\#&>y                                Lengths of groups of blocks
1 3 7 15 31

   ]p=: >_2{y                             A typical block
0 2 4 6 8 10 12 14

   ]q=: >_1{y                             Its successor
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30

   (2 * p +/ 0 1);(q=,2*p+/0 1)           Relation between blocks
┌─────┬───────────────────────────────┐
│ 0  2│1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1│
│ 4  6│                               │
│ 8 10│                               │
│12 14│                               │
│16 18│                               │
│20 22│                               │
│24 26│                               │
│28 30│                               │
└─────┴───────────────────────────────┘
```

The relations between the successive blocks `p` and `q` observed above lead to an alternative recursive definition of `jr`:

```
   (jar=: 0:`(2: * jar@<:@>.@-: + 2&|@<:)@.*"0) i. 6
0 0 2 0 2 4 6 0 2 4 6 8 10 12 14 0
```

Since each block of `jr` ascends in steps of two, since each re-starts at zero, and since the lengths of groups of blocks are as illustrated above, a non-recursive solution (analogous to GKP1.9) may be defined as follows:

```
   jnr=: 2:*1:+]-(2:^<.@(2&^.))@>:
   jnr i. 16
0 0 2 0 2 4 6 0 2 4 6 8 10 12 14 0
```

To compare one of the functions developed here with the corresponding function in GKP, it is necessary (because of the use of zero-origin indexing), to apply it *under decrement*; that is, decrement the argument, apply the function, and then apply the inverse function *increment*. Thus:

```
   >:@jnr@<: 100
73

   jnr&.<: 100                 Dual of jnr with respect to decrement
73
```

D. QUICKSORT

Chapter 2 of GKP (page 28) provides a recursive statement of the number of comparisons needed in Hoare's Quicksort algorithm. We will conclude this chapter with a recursive definition of the quicksort algorithm itself:

```
   sel=: 1 : '] #~ ] x. {.'
   qsort=: ]`($:@(<sel), =sel, $:@(>sel)) @. (1:<#)
   qsort y =: 15 2 9 10 4 0 13 13 18 7
0 2 4 7 9 10 13 13 15 18
   y /: y
0 2 4 7 9 10 13 13 15 18
```

E. NOTATION

Fork. The three verbs in the definition `mean=: +/ % #` form a *fork*, and `mean x` is equivalent to `(+/x) % (#x)`. The fork phrase must be isolated; that is, `(+/ % #) x` gives the mean of `x`, but `+/ % # x` does not.

Atop. The conjunction `@` first appears in the expression `] * f@<:`. The phrase `f@<:` produces a function equivalent to applying `f` atop (that is, to the result of) the decrement function `<:`.

Curtail. The function `}:` introduced in the section on Josephus *curtails* its argument, dropping the last item or *tail*, itself selected by the function `{:`. Similarly, `}.` beheads its argument , and `{.` selects its head.

Cut. In the section on Josephus, the phrase `<;.1` is applied to the ravelled argument `,J\Ai 16` to box (<) intervals of the argument demarked by occurrences of the head of the argument, which therefore serves as a *fret*. The *cut* conjunction `;.` may be used with functions other than box; for example, `+/;.1` applied to the same argument yields `1 4 16 32 1`.

The right argument of `;.` concerns the fret; using the head if it is `1`, the tail if it is `2`, and excluding the fret itself from the intervals if it is negative. Thus `+/;._1` applied to the same argument would yield `0 3 15 31 0`.

Erasure. If the names in the phrase ``b`(p,.q,.r)@.c`` that begins the recursive definition for the Hanoi problem were pre-defined to be other than verbs it would not work as expected. It is prudent to precede such a phrase by one that erases names that have not yet been assigned their intended referents. Use:

`erase=: 4!:55@;:` , as in `erase 'b p q r'`

Rank. `f"0 1` applies `f` between each rank `0` element (atom) of its left argument and each rank `1` element (vector) of its right. For example:

```
   1 2 3 (,"0 1 ; ,"1 0 ; ,"1 1 ; ,"1) 4 5 6
```

1 4 5 6 2 4 5 6 3 4 5 6	1 2 3 4 1 2 3 5 1 2 3 6	1 2 3 4 5 6	1 2 3 4 5 6

Scans. The adverb `\` (first used in the expression `2 +/\ a`) applies to its verb argument `+/` to produce a verb that applies `+/` (summation) to each length-2 (as specified by the left argument) infix of the right argument `a`, therefore producing sums over all adjacent pairs of `a`. The monadic case `+/\ a` applies `+/` to each prefix of `a`, and therefore produces subtotals or partial sums. Similarly, `*/\` produces partial products, and `<./\` produces partial minima.

The adverb `\.` applies its argument to suffixes, and the adverb `/.` applies its argument to oblique lines of a table. For example:

```
   c=: 1 2 1 [ d=: 1 3 3 1
   (c */ d) ; (+//. c */ d) ; <(</. c */ d)
```

```
┌───────┬─────────────────┬──────────────────────────┐
│1 3 3 1│1 5 10 10 5 1    │┌─┬─────┬─────┬─────┬───┬─┐│
│2 6 6 2│                 ││1│3 2  │3 6 1│1 6 3│2 3│1││
│1 3 3 1│                 │└─┴─────┴─────┴─────┴───┴─┘│
└───────┴─────────────────┴──────────────────────────┘
```

Symmetry. The proof in the section on triangular numbers uses the phrase "`+/` is symmetric". A monadic function is said to be symmetric if any permutation of its argument yields the same result. The function `+/` is symmetric because the (dyadic) function `+` is both commutative and associative.

Self-reference. The primitive `$:` provides self-reference to a function being defined, as in the recursive definition in the introduction to this chapter. It is also used to refer to the other case (monadic or dyadic) of a function being defined by the conjunction `:`, as illustrated by the definition of the ambivalent function `j` in the Josephus problem.

Under. `f&.g` applies `f` to the result of `g`, and then applies the inverse of `g` to that result:

```
   (+: ; *: ; *:&.+:) 0 1 2 3 4          Double, square, square under double
┌─────────┬──────────┬───────────┐
│0 2 4 6 8│0 1 4 9 16│0 2 8 18 32│
└─────────┴──────────┴───────────┘
```

The function `f&.g` is sometimes referred to as the *dual* of `f` *with respect to* `g`.

Chapter 2

SUMS

A. NOTATION

GKP2.1 introduces sums of the form $a_1+a_2+...+a_n$ "where each a_k is a number that has been defined somehow". We will therefore treat `a` as a function, and the list of indices `k` as a second function, typically `i.` or the function `Ei=: i.@>:`. For example:

```
   Ei=: i.@>:                          Extended indices

   Ei 5
0 1 2 3 4 5

   a=: *:                              Square

   a Ei 5
0 1 4 9 16 25

   +/ a Ei 5
55
```

We will define an adverb `S` such that `f S x` yields the sum `+/ f x`. Thus:

```
   S=: (+/@:)(@Ei)("0)

   ] S
+/@:]@Ei"0

   ] S 5
15
   x=: Ei 10
   ] S x                               Triangular numbers
0 1 3 6 10 15 21 28 36 45 55
   *: S x                              Sums of squares 0-10
0 1 5 14 30 55 91 140 204 285 385
```

If `p` is a *proposition* that yields `1` or `0`, then `(f*p)S` yields the sum of `f` over those indices that satisfy the proposition. For example:

```
   Ispr=: (1:=#@q:) :: 0:"0            Proposition is prime

   Ispr Ei 15                          Prime test
0 0 1 1 0 1 0 1 0 0 0 1 0 1 0 0

   (%*Ispr)S Ei 7                      Sum of reciprocals of primes
0 0 0.5 0.8333333 0.8333333 1.03333 1.03333 1.17619
```

Alternatively, the function `f*p` can be replaced by a function that selects only those arguments that satisfy a proposition. Thus:

```
 (Ispr # ]) Ei 7
2 3 5 7

   %@(Ispr#]) S  7
1.17619

   %@(Ispr#]) S Ei 7
0 0 0.5 0.8333333 0.8333333 1.03333 1.03333 1.17619

   4 5$ %@(Ispr#]) S Ei 19
      0       0     0.5 0.8333333 0.8333333
1.03333 1.03333 1.17619   1.17619   1.17619
1.17619  1.2671  1.2671   1.34402   1.34402
1.34402 1.34402 1.40285   1.40285   1.45548
```

B. SUMS AND RECURRENCES

As stated in GKP2.6, the sum `a S` is equivalent to the following recursively defined function:

```
   a=: *:
   sum=: a`(a+$:@<:) @. * "0
   sum Ei 10                                    Sum of squares
0 1 5 14 30 55 91 140 204 285 385
   a=: ]
   sum Ei 10                                    Triangular numbers
0 1 3 6 10 15 21 28 36 45 55
   a=: 3:+2:*]
   a Ei 10
3 5 7 9 11 13 15 17 19 21 23
   sum Ei 10
3 8 15 24 35 48 63 80 99 120 143
   a S Ei 10
3 8 15 24 35 48 63 80 99 120 143
```

In a manner analogous to the *repertoire* method of GKP, we will find a non-recursive equivalent to a recursively defined function by finding a polynomial fit to a few of its values. To this end we will use the adverb:

```
   CPA=: (@Ei) %. ^/~@Ei
```

so defined that `f CPA n` yields the coefficients of a polynomial approximation of order `n` to the function `f`. For example:

```
   a=: *:
   a CPA
a@Ei %. ^/~@Ei
   sum i=: Ei 7
0 1 5 14 30 55 91 140
   ]c=: sum CPA 4                    Coeffs of polynomial equivalent of sum
0 0.1666667 0.5 0.3333333 0
   c p. i                            Test of coefficients
0 1 5 14 30 55 91 140
   6*c
0 1 3 2 0
   (0 1 3 2%6) p. i                  Alternate expressions of polynomial
0 1 5 14 30 55 91 140
   (i + (3*i^2) + (2*i^3))%6
0 1 5 14 30 55 91 140
   a=: ^&3
   sum i                             Sum of cubes
0 1 9 36 100 225 441 784
   d=: sum CPA 5
   d
0 0 0.25 0.5 0.25 0
   d p. i
0 1 9 36 100 225 441 784
   (*:i)*(*:i+1)%4
0 1 9 36 100 225 441 784
   (*:i*i+1)%4
0 1 9 36 100 225 441 784
```

C. MANIPULATION OF SUMS

A monadic function `g` is said to be *symmetric* if it is invariant under any permutation of its argument; that is, `g=g@p` for any permutation function `p` . For example:

```
   p=: 3 1 0 4 2&{
```

```
   p 'ABCDE'
DBAEC
   g=: */                                    Product over
   (g ; p ; g@p) x=: 2 7 8 1 8
┌───┬─────────┬───┐
│896│1 7 2 8 8│896│
└───┴─────────┴───┘
```

The function `*/` is symmetric because the dyadic function `*` (multiplication) is both associative and commutative. In general, it is easy to prove that `f/` is symmetric if `f` is both associative and commutative. In particular, summation (`+/`) is symmetric.
The relations expressed by GKP2.5-7 are re-expressed in the following tautologies, using `c` to denote a constant scalar function, `p` to denote a permutation, and the adverb `S` defined in §A:

```
   a=: 3 2&p. [. b=: *: [. c=: 0.1"0 [. p=: 97&A. Example functions
   t1=: (c*a) S = c*a S
   t2=: (a+b) S = a S + b S
   t3=:     a S = a@p S
   (t1 5),(t2 5),(t3 5)
1 1 1
   ((c*a) S ; a S ; c * a S) 5
┌───┬──┬───┐
│4.8│48│4.8│
└───┴──┴───┘
   ((a+b) S ; a  S ; b  S; a  S + b  S) 5
┌───┬──┬──┬───┐
│103│48│55│103│
└───┴──┴──┴───┘
   (a S ; p@Ei ; a@p S) 5
┌──┬───────────┬──┐
│48│0 5 1 2 4 3│48│
└──┴───────────┴──┘
```

As shown in GKP, these laws can be used to justify the method of Gauss for expressing a triangular numberas a product (used earlier in Chapter 1 of this text). We will illustrate this as follows, noting that reversal (`|.`) is a permutation:

```
   ((] S);(|. S);(-:@(]+|.) S);(]+|.)@Ei) 4
┌──┬──┬──┬─────────┐
│10│10│10│4 4 4 4 4│
└──┴──┴──┴─────────┘
```

Partitioning of the type expressed by GKP2.19 may be illustrated as follows:

```
   se=: e#] [. so=: o#] [. e=: -. @ o=: 2&|
   (e;o;se;so) Ei 4                          Even, Odd, Select even, Select odd
┌─────────┬─────────┬─────┬───┐
│1 0 1 0 1│0 1 0 1 0│0 2 4│1 3│
└─────────┴─────────┴─────┴───┘
   ((se S);(so S);((se S)+(so S));(] S))4
┌─┬─┬──┬──┐
│6│4│10│10│
└─┴─┴──┴──┘
```

Other splitting of sequences used in the last part of §2.3 of GKP can be effected by the head, behead, tail, and curtail functions (`{. }. {: }:`) used in Chapter 1 of this text, or by functions such as take and drop (`3&{.` and `3&}.`, etc.). More general cutting is provided by the dyadic case of the function produced by the *cut* conjunction, in which the *ones* in the boolean left argument mark the cut points. For example:

```
   u=: 1 0 0 1 0 0 1 0
   ] v=: i. # u
```

```
0 1 2 3 4 5 6 7
   u <;.1 v
┌─────┬─────┬───┐
│0 1 2│3 4 5│6 7│
└─────┴─────┴───┘
   u +/;.1 v
3 12 13
   u <;.2 v
┌─┬─────┬─────┐
│0│1 2 3│4 5 6│
└─┴─────┴─────┘
```

The marginal note on page 26 of GKP concerning an approximation to π may be expressed as follows:

```
   (8: % ((4:*]) + 1:)*((4:*]) + 3:)) S 1000
3.14109
   (8: % 1 4&p. * 3 4&p.) S 1000              Equivalent use of polynomials
3.14109
   d=: 8 %~ c=:+//.1 4 */ 3 4
   c;((8: % c&p.) S 1000);d;(%@(d&p.) S 1000)
┌───────┬───────┬─────────┬───────┐
│3 16 16│3.14109│0.375 2 2│3.14109│
└───────┴───────┴─────────┴───────┘
```

D. MULTIPLE SUMS

A matrix or table is said to have two indices (or two axes) because its rows and columns may be selected independently; summation can be applied over either index to produce a vector or list result, which may again be summed.

More generally, an *array* or *report* may have `n` axes, and summation may be applied over any one of them by using a sum of appropriate rank. For example:

```
   r=: i. 4 3 2
   (] ; $ ; # ; #@$ ; +/"0 ; +/"1 ; +/"2 ; +/"3 ; +/) r
┌─────┬─────┬─┬─┬─────┬────────┬─────┬─────┬─────┐
│ 0  1│4 3 2│4│3│ 0  1│ 1  5  9│ 6  9│36 40│36 40│
│ 2  3│     │ │ │ 2  3│13 17 21│24 27│44 48│44 48│
│ 4  5│     │ │ │ 4  5│25 29 33│42 45│52 56│52 56│
│     │     │ │ │     │37 41 45│60 63│     │     │
│ 6  7│     │ │ │ 6  7│        │     │     │     │
│ 8  9│     │ │ │ 8  9│        │     │     │     │
│10 11│     │ │ │10 11│        │     │     │     │
│     │     │ │ │     │        │     │     │     │
│12 13│     │ │ │12 13│        │     │     │     │
│14 15│     │ │ │14 15│        │     │     │     │
│16 17│     │ │ │16 17│        │     │     │     │
│     │     │ │ │     │        │     │     │     │
│18 19│     │ │ │18 19│        │     │     │     │
│20 21│     │ │ │20 21│        │     │     │     │
│22 23│     │ │ │22 23│        │     │     │     │
└─────┴─────┴─┴─┴─────┴────────┴─────┴─────┴─────┘
```

Repeated summation will eventually produce a single (scalar) result and, because summation is symmetric, this result will be the same whatever the order of summation. Moreover, the same is true of a table formed as the outer product of two vectors, and indeed of any permutation of such a table. For example:

```
   (+/+/+/r),(+/+/+/"2 r),(+/"1+/"1+/"1 r)
276 276 276
   V=: 2 4 6
   W=: 3 1 4 1
   t=: V */ W
   p=: 4 & A.
```

```
   p 0 1 2
2 0 1
   t;(+/+/t);(+/+/"1 t);(p t);(+/+/p t)
┌─────────┬───┬───┬─────────┬───┐
│ 6 2  8 2│108│108│18 6 24 6│108│
│12 4 16 4│   │   │ 6 2  8 2│   │
│18 6 24 6│   │   │12 4 16 4│   │
└─────────┴───┴───┴─────────┴───┘
```

The symmetry of summation likewise ensures that complete summation of a product table `V */ W` remains unchanged if the arguments `V` and `W` are permuted. Thus:

```
   (+/,V*/W);(p V);(|.W);((p V)*/|.W);(+/,(p V) */|.W)
┌───┬─────┬───────┬─────────┬───┐
│108│6 2 4│1 4 1 3│6 24 6 18│108│
│   │     │       │2  8 2  6│   │
│   │     │       │4 16 4 12│   │
└───┴─────┴───────┴─────────┴───┘
```

The following results illustrate GKP2.28:

```
   (V*/W);(+/+/V*/W);(+/V);(+/W);((+/V)*(+/W))
┌─────────┬───┬──┬─┬───┐
│ 6 2  8 2│108│12│9│108│
│12 4 16 4│   │  │ │   │
│18 6 24 6│   │  │ │   │
└─────────┴───┴──┴─┴───┘
```

More generally, it is convenient to treat tables as outer products of functions that apply to lists of integers. For example:

```
   a=: 1: + 2: * ] [. b=: *:

   t=: a */ b


   (];a;b;t;(+/@(+/@t));(+/@(+/"1@t))) Ei 4
┌─────────┬─────────┬──────────┬─────────────┬───┬───┐
│0 1 2 3 4│1 3 5 7 9│0 1 4 9 16│0 1  4  9  16│750│750│
│         │         │          │0 3 12 27  48│   │   │
│         │         │          │0 5 20 45  80│   │   │
│         │         │          │0 7 28 63 112│   │   │
│         │         │          │0 9 36 81 144│   │   │
└─────────┴─────────┴──────────┴─────────────┴───┴───┘
```

Propositions may be used to limit summation to subsets. For example:

```
   pr=: <:/~                                Upper triangle
    (pr;(pr * t);(+/@,@(pr*t))) Ei 4
┌─────────┬─────────────┬───┐
│1 1 1 1 1│0 1  4  9  16│584│
│0 1 1 1 1│0 3 12 27  48│   │
│0 0 1 1 1│0 0 20 45  80│   │
│0 0 0 1 1│0 0  0 63 112│   │
│0 0 0 0 1│0 0  0  0 144│   │
└─────────┴─────────────┴───┘
```

The adverb `S` of §A may be used to illustrate the two limbs of GKP2.33 as follows:

```
   (,@(pr * a */ a) S ; 2: %~ *:@(a S) + *:@a S) 4
┌───┬───┐
│395│395│
└───┴───┘
```

The left limb sums all elements of the upper triangle of the table `a */ a`; the right halves the square of the sum of `a` added to the sum of its squares.

E. GENERAL METHODS

In §B we developed a general method (analogous to the repertoire method of GKP) which determined the coefficients of a polynomial equivalent to the sum over a specified function. In particular, we treated the cases of squares and cubes.

There are several reasons why polynomials are useful in exploring expressions for sums:

A) A simple function `Epa` may be used to expand a polynomial `f=: c&p.`; that is, to obtain a polynomial `g=: d&p.` such that `g x` equals `f x+1`. This is clearly useful in developing recursion and inductive proofs.

```
   Epa=: Bc@# X ]
     X=: +/ . *                      Matrix product
     Bc=: i. !/ i.                   Binomial coefficients
   ]d=: Epa c=: 1 2 3
6 8 3
   (c p. x+1) ,: d p. x=: 0 1 2 3
6 17 34 57.
6 17 34 57
```

B) It is easy to obtain sums and products of functions expressed as polynomials. For example, `c&p. * d&p.` is equivalent to `(cáppád)&p.`, where `pp` is a polynomial product function. Thus:

```
   pp=: +//.@(*/)
   1 2 1 pp 1 3 3 1
1 5 10 10 5 1
    0 1 pp 1 1 pp 1 2
0 1 3 2
  pp/0 1,1 1,:1 2
0 1 3 2
```

C) The adverb `CPA` of §B used in `c=: f CPA d` yields the coefficients of a polynomial approximation of degree `d` to the function `f`. Thus:

```
   CPA=: (@Ei) %. ^/~@Ei
   ]c=:  %: CPA 6
0 1.71544 _1.0635 0.43653 _0.0995919 0.011677 _0.00054824
   c p. x=: Ei 5
0 1 1.41421 1.73205 2 2.23607
   %: x
0 1 1.41421 1.73205 2 2.23607
```

D) A linear function of a collection of polynomials is equivalent to a polynomial whose coefficients are the same linear function of their coefficients. For example, using the matrix product `X=: +/ . *`:

```
   C=: 3 1 2 , 0 1 2 3 ,: 2 1
   d=: 0 1 4 [ y=: 0 1 2 3 4
   C;(C p./ y);(d X C p./ y);((d X C) p. y)
```

3 1 2 0 0 1 2 3 2 1 0 0	3 6 13 24 39 0 6 34 102 228 2 3 4 5 6	8 18 50 122 252	8 18 50 122 252

E) The derivative and the integral of a polynomial `c&p.` are also polynomials:

`(1: }. ] * i.@#)@[ p. ]` and `(0: , ] % >:@i.@#)@[ p. ]`

Method 5 on page 46 of GKP may be paraphrased by the following tautology:

```
   t46=: +/@*:@Ai = +/@(+/\.)@Ai
   t46"0 i. 6
1 1 1 1 1 1
```

The right limb of `t46` may be illustrated as follows:

```
   ] i=: Ai t=: 4
1 2 3 4
   (<\.i);(+/\.i);(+/+/\.i)
┌──────────────────────┬─────────┬──┐
│┌───────┬─────┬───┬─┐│10 9 7 4│30│
││1 2 3 4│2 3 4│3 4│4││         │  │
│└───────┴─────┴───┴─┘│         │  │
└──────────────────────┴─────────┴──┘
```

It may also be paraphrased by an expression that multiplies a table of integers by a boolean upper triangle (to suppress the elements suppressed by the foregoing suffix scan) before performing a final summation:

```
   A=: (t,t)$1
   I=: +/\. A
   U=: <:/~ i.@#I
   A;I;U;(I*U);(+/"1 I*U);(+/+/"1 I*U)
┌───────┬───────┬───────┬───────┬──────────┬──┐
│1 1 1 1│4 4 4 4│1 1 1 1│4 4 4 4│16 9 4 1│30│
│1 1 1 1│3 3 3 3│0 1 1 1│0 3 3 3│          │  │
│1 1 1 1│2 2 2 2│0 0 1 1│0 0 2 2│          │  │
│1 1 1 1│1 1 1 1│0 0 0 1│0 0 0 1│          │  │
└───────┴───────┴───────┴───────┴──────────┴──┘
```

F. FINITE CALCULUS

The difference operator of GKP2.42 and the falling factorial function of GKP2.43 may be defined as follows:

```
   fd=: 1 : 'x.@>: - x.'          Forward difference adverb
   ff=: */@([ - i.@])"0           Falling factorial function
```

For example:

```
   *: fd
*:@>: - *:
   ^&3 fd
^&3@>: - ^&3
    ((^&2 fd);(^&3 fd);(^&4 fd)) x=: 0 1 2 3 4
┌─────────┬────────────┬────────────────┐
│1 3 5 7 9│1 7 19 37 61│1 15 65 175 369│
└─────────┴────────────┴────────────────┘
   (3*x^2)+(3*x)+1
1 7 19 37 61
    4 ff x
1 4 12 24 24
   x ff/x
1 0  0  0  0
1 1  0  0  0
1 2  2  0  0
1 3  6  6  0
1 4 12 24 24
```

The function `^!.r` is a *variant* of the power function defined by the expression:

```
   g=: */@([ + r"_ * i.@])"0
```

In particular, `^!._1` is equivalent to the falling factorial `ff` defined above, `^!.0` is the power function itself, and `^!.1` is the *rising* factorial. Moreover, the parameter `r` is not restricted to integers.

Similarly, `p.!.r` is a variant of the polynomial defined as a weighted sum of the function `^!.r` rather than `^`. In particular, `p.!.0` is equivalent to `p.`, and `p.!._1` is a polynomial based on the falling factorial. For example:

```
   ff=: ^!._1 [. fp=: p.!._1
```

```
   c=: 2 3 1 4 [ d=: 2 8 13 4 [ x=: 4 5 6
   (x ^ 5);(x ff 5);(c p. x);(d fp x)
┌──────────────┬─────────┬───────────┬───────────┐
│1024 3125 7776│0 120 720│286 542 920│286 542 920│
└──────────────┴─────────┴───────────┴───────────┘
```

Because the derivative of `^&t` is `t` times `^&(t-1)`, the derivative of the polynomial `^&t` is the polynomial `(Dpc c)&p.` and its integral is `(Ipcác)&p.`, where `Dpc` and `Ipc` are defined as follows:

```
   Dpc=: 1: }. ] * i.@#
   Ipc=: 0: , ] % >:@i.@#
```

For example:

```
   c&p. x=: 0 1 2 3 4
2 10 44 128 286
   c&p. D. 1 x                    First derivative of c&p.
3 17 55 117 203
   (Dpc c)&p. x
3 17 55 117 203
   (Ipc c)&p. x
0 4.83333 28.6667 109.5 309.333
   (Ipc c)&p. D. 1 x
2 10 44 128 286
```

As stated in GKP2.45, the difference `ff&m fd` is equivalent to `m` times `ff&(m-1)`. For example:

```
   m=: 4
   ((ff&m fd);(ff&(m-1));(m"_ * ff&(m-1))) x
┌───────────┬──────────┬───────────┐
│0 0 0 24 96│0 0 0 6 24│0 0 0 24 96│
└───────────┴──────────┴───────────┘
```

Thus the *difference* of the falling polynomial behaves analogously to the *derivative* of the ordinary polynomial:

```
   (c&fp ; c&fp fd ; (Dpc c)&fp) x
┌────────────┬────────────┬────────────┐
│2 5 10 41 122│3 5 31 81 155│3 5 31 81 155│
└────────────┴────────────┴────────────┘
   ((Ipc c)&fp fd ; +/\@(c&fp)) x
┌────────────┬────────────┐
│2 5 10 41 122│2 7 17 58 180│
└────────────┴────────────┘
```

Expecting that the integral might be related to sums over the falling polynomial, we compare them as follows:

```
   (((Ipc c)&fp);(+/\@(c&fp))) x
┌───────────┬────────────┐
│0 2 7 17 58│2 7 17 58 180│
└───────────┴────────────┘
```

However, in order to apply the results of finite differences and integrals to ordinary polynomials we must develop a transformation `t` such that `(t c)&fp` is equivalent to `c&p.`. To this end we express a polynomial as a linear function of its coefficients.

If `vm` is the table of powers `x ^/ i.#c` (called a Vandermonde matrix), then `vm X c` is equivalent to `c&p. x`. An analogous matrix may be defined for the falling factorial function. Thus:

```
   X=: +/ . *                    Matrix product
   vm=: x ^/ i.#c
   vmf=: x ^!._1/ i.#c
   ,.&.>(vm;(vm X c);(c p.x);vmf;(vmf X c);c fp x)
┌─────────┬───┬───┬─────────┬───┬───┐
```

```
│1 0  0  0│  2│  2│1 0  0  0│  2│  2│
│1 1  1  1│ 10│ 10│1 1  0  0│  5│  5│
│1 2  4  8│ 44│ 44│1 2  2  0│ 10│ 10│
│1 3  9 27│128│128│1 3  6  6│ 41│ 41│
│1 4 16 64│286│286│1 4 12 24│122│122│
└─────────┴───┴───┴─────────┴───┴───┘
```

The transformation `t` may therefore be defined as the “quotient” of square Vandermonde matrices using `i.#c` for `x`:

```
   s=: i. (^/ %. ff/) i.
   t=: s@# X ]
   d=: t c
   ,.&.>((s 4);(%.s 4);c;d;(c p. x);(d p.!._1 x))
┌───────┬──────────┬─┬──┬───┬───┐
│1 0 0 0│1 0  0  0 │2│ 2│  2│  2│
│0 1 1 1│0 1 _1  2 │3│ 8│ 10│ 10│
│0 0 1 3│0 0  1 _3 │1│13│ 44│ 44│
│0 0 0 1│0 0  0  1 │4│ 4│128│128│
│       │          │ │  │286│286│
└───────┴──────────┴─┴──┴───┴───┘
```

The transformation `t` may now be used to define a transformation from the coefficients of the polynomial `+/\@(c&p.)`:

```
   st=: Epa@(t^:_1)@Ipc@t
     Ipc=: 0: , ] % >:@i.@#
     Epa=: Bc@# X ]
       X=: +/ . *
       Bc=: i. !/ i.
   e=: st c
   ,.&.>(c;e;(+/\c p. x);(e p. x))
┌─┬───────┬───┬───┐
│2│      2│  2│  2│
│3│3.66667│ 12│ 12│
│1│      3│ 56│ 56│
│4│2.33333│184│184│
│ │      1│470│470│
└─┴───────┴───┴───┘
```

The polynomial coefficients of the successive powers appear as the successive rows of the identity matrix, and the function `st` may now be applied to them to obtain a table of coefficients of sums of powers:

```
   (Id=: i. =/ i.) 5              Identity matrix function
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
   st"1 Id 5
           1           1            0         0    0   0
           0         0.5          0.5         0    0   0
_5.55112e_17   0.1666667          0.5 0.3333333    0   0
           0           0         0.25       0.5 0.25   0
 7.21645e_16 _0.03333333 _8.88178e_16 0.3333333  0.5 0.2
```

Tiny values of the order of `1e_17` that appear in this result should be treated as zeros; they arise from the limited precision of the computer calculations. They may be suppressed from the display by the following *threshold* function as shown:

```
   Thr=: ] * 0.1&^@[ <: |@]
   3 Thr st"1 Id 5
1           1    0         0    0   0
0         0.5  0.5         0    0   0
0   0.1666667  0.5 0.3333333    0   0
0           0 0.25       0.5 0.25   0
0 _0.03333333    0 0.3333333  0.5 0.2
```

The function `s` used in the definition of the transformation `t` is simply related to the Stirling numbers discussed in Chapter 6:

```
   (|: s 7) ; (|: | %. s 7)
┌────────────────────┬──────────────────────────┐
│1 0  0  0  0  0 0│1   0   0   0  0  0 0│
│0 1  0  0  0  0 0│0   1   0   0  0  0 0│
│0 1  1  0  0  0 0│0   1   1   0  0  0 0│
│0 1  3  1  0  0 0│0   2   3   1  0  0 0│
│0 1  7  6  1  0 0│0   6  11   6  1  0 0│
│0 1 15 25 10  1 0│0  24  50  35 10  1 0│
│0 1 31 90 65 15 1│0 120 274 225 85 15 1│
└────────────────────┴──────────────────────────┘
```

The extension of the rising factorial to negative exponents (GKP2.51) uses `m` factors; from `x` to `x+m-1` or, for the case of `-m`, from `x+1` to `x+m`. The final result is their product, or, in the case of `-m`, its reciprocal. We will define more general increments that provide specification of the step size, thus extending the definition to falling factorials (with a step size of `_1`) as well:

```
   inc=: [ * i.@|@] + 0: > ]

   1 _1 inc"0/ 5 _5
 0  1  2  3  4
 1  2  3  4  5

 0 _1 _2 _3 _4
_1 _2 _3 _4 _5
```

Finally, we define an adverb whose left argument specifies the step size:

```
   FAC=: 1 : '*/@([ + x."0 inc ]) ^ *@]'
   x=: 4 5 6 [ m=: 3 _3

   (x 1 FAC"0/ m) ; (x _1 FAC"0/ m)
┌───────────────┬──────────────┐
│120 0.004761905│ 24  0.1666667│
│210  0.00297619│ 60 0.04166667│
│336 0.001984127│120 0.01666667│
└───────────────┴──────────────┘
```

G. INFINITE SUMS

The expression `+/ f i. n` sums the function `f` over the first `n` integers, but it cannot be used directly to sum only until some condition (such as a limiting value) has been reached. For this we will define an adverb `step` such that the function `f step` performs a single step in the summation. Thus:

```
   step=: 1 : '>:@{. , {: + x.@{.'

   *: step
>:@{. , {: + *:@{.

   k=: i. 8

   (*: step ^: k 0 0) ; (3: ^ -) step ^: k 0 0
┌────┬─────────┐
│0  0│0       0│
│1  0│1       1│
│2  1│2 1.33333│
│3  5│3 1.44444│
│4 14│4 1.48148│
│5 30│5 1.49383│
│6 55│6 1.49794│
│7 91│7 1.49931│
└────┴─────────┘
```

Two further adverbs serve to test whether the sum is still changing (has not converged), and to apply the step until it does converge:

```
   test=:1 : '{:@([ ~: x. step)'
```

```
   lim=: 1 : '{:@(x. step^:(x. test)^:_)'
   (3: ^ -) lim 0 0
1.5
```

H. NOTATION

Derivatives. The conjunction `D.` is used as in `f D. k` to produce the `k`th derivative of the function `f`. The adverb `D1=: ("0)(D.1)` produces the scalar first derivative.

The property of the falling factorial stated in GKP2.45 is used in Iverson [4] as a basis from which to derive the falling factorial as a function for which differencing is analogous to differentiation of the power functions.

Matrix quotient. The monadic case of `%.` is the matrix inverse, and the dyadic case may be called the matrix quotient: `m%.t` is defined as `(%.t) X m`. If `t` is a tall (and therefore singular) matrix, the result is a best fit in the least-squares sense.

Oblique. The oblique adverb `/.` applies its function argument to each of the (forward-sloping) diagonals of a matrix argument of the resulting verb. For example:

```
]m=: 1 2 1 */ 1 3 3 1
1 3 3 1
2 6 6 2
1 3 3 1
   </. m
```

1	3 2	3 6 1	1 6 3	2 3	1

```
   +//. m
1 5 10 10 5 1
```

Passive and reflexive. The sentence `a f~ b` applies `f` *passively* (commuting the arguments), and `f b` applies it *reflexively* (as in `b f b`). For example:

```
   into=: %~
   10 into 0 1 2 3 4
0 0.1 0.2 0.3 0.4
   (*~ 4) , (4*4)
16 16
```

Stopes. `^!.r` is a *variant* of the power function defined by the fact that `x ^!.r t` is equivalent to `*/xá+ár * i.t`. Special cases are the falling factorial (`^!._1`), the rising factorial (`^!.1`), and the power function itself (`^!.0`). The function `p.!.r` is a similar variant of the polynomial. These variants may be used instead of the functions `ff` and `fp` defined in this chapter, and `^!._1` is so used in §D of Chapter 5.

Prime factors. The function `q:` used in the proposition `Ispr=: 1:=#@q:` produces the list of prime factors of its argument; that is, `]=*/@q:` is a tautology. The function `p:` produces primes. For example:

```
   p: 0 1 2 3 4 5
2 3 5 7 11 13
   ]q=: p:^:_1 (2^31)-1
105097564
   p: q
2147483647
```

Chapter 3
INTEGER FUNCTIONS

A. FLOOR AND CEILING

As stated in GKP, the *floor* or *integer part* of a real number is the greatest integer that does not exceed it. For example:

```
   Si=: Ei@+: - ]              Symmetric integers
     Ei=: i.@>:                Extended integers
   Si 6
_6 _5 _4 _3 _2 _1 0 1 2 3 4 5 6
   x=: 5%~ Si 6
   < (>. , ] ,: <.) x
```

```
  _1 _1    0    0    0    0 0   1   1   1   1 1   2
_1.2 _1 _0.8 _0.6 _0.4 _0.2 0 0.2 0.4 0.6 0.8 1 1.2
  _2 _1   _1   _1   _1   _1 0   0   0   0   0 1   1
```

We will use the floor and ceiling to illustrate the definition of *propositions* (to yield `1` if the argument satisfies certain criteria) and of *tautologies* (to yield `1` for any argument). Thus:

```
   IsI=: ] = <.                Is integral
   IsMi=: ] -: /:~             Is monotone increasing (matches sort)
   IsMd=: ] -: \:~             Is monotone decreasing
   GKP3_3=: IsMi@(<: , <. , ] , >. , >:)
   GKP3_4=: <. = >.&.-
   GKP3_6=: IsI@] <: <.@+ = <.@[ + <.@]
   IsI x
0 1 0 0 0 0 1 0 0 0 0 1 0
   IsMi <. x      Floor of a monotone increasing argument is monotone increasing
1
```

The function `GKP3_3` is so named because it represents GKP3.3; `GKP3_4` states that floor is the dual of ceiling with respect to arithmetic negation. Since duality is symmetric, ceiling is also the dual of floor. The function `GKP3_6` may be read as: The right argument being an integer implies that the floor of the sum equals the floor of the left argument added to the floor of the right.

The dyadic function `<:` (*less than or equal* or *does not exceed*) represents implication because the "false" and "true" resulting from relations and from boolean functions are represented by `0` and `1` in the manner used by Boole himself. For example:

```
   x GKP3_6"0/ x
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 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 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 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 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 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 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1
```

Because the phrase `<.@[ + <.@]` is equivalent to `+&<.`, and because implication becomes redundant if the right argument `]` is replaced by the integer `<.@]`, the tautology may also be written in simpler forms as follows:

```
   GKP3_6a=: (IsI@] <: <.@+ = +&<.)"0
   GKP3_6b=: (<.@([ + <.@]) = +&<.)"0
```

This last definition may be read as "The floor of the sum of one argument with the floor of the other equals the sum of their floors".

The function `GKP3_3` asserts that decrement, floor, identity, ceiling, and increment are in non-decreasing order. Alternatively this may be stated by asserting that successive pairs are each in non-decreasing order:

```
   GKP3_3a=: *./@(2: <:/\ (<: , <. , ] , >. , >:))
   GKP3_3a"0 x
1 1 1 1 1 1 1 1 1 1 1 1 1
```

To examine the definitions of floor and ceiling on complex numbers, we use a non-negative table of them as follows:

```
    <xm=: j./~ 5%~ Ei 5
┌─────────────────────────────────────────┐
│  0   0j0.2   0j0.4   0j0.6   0j0.8   0j1│
│0.2 0.2j0.2 0.2j0.4 0.2j0.6 0.2j0.8 0.2j1│
│0.4 0.4j0.2 0.4j0.4 0.4j0.6 0.4j0.8 0.4j1│
│0.6 0.6j0.2 0.6j0.4 0.6j0.6 0.6j0.8 0.6j1│
│0.8 0.8j0.2 0.8j0.4 0.8j0.6 0.8j0.8 0.8j1│
│  1   1j0.2   1j0.4   1j0.6   1j0.8   1j1│
└─────────────────────────────────────────┘

   (<. ; >.) xm
┌──────────────────────┬──────────────────────┐
│0 0 0   0   0 0j1     │0 0j1 0j1 0j1 0j1 0j1 │
│0 0 0   0 0j1 0j1     │1 0j1 0j1 0j1 0j1 1j1 │
│0 0 0 0j1 0j1 0j1     │1   1 0j1 0j1 1j1 1j1 │
│0 0 1   1 0j1 0j1     │1   1   1 1j1 1j1 1j1 │
│0 1 1   1   1 0j1     │1   1 1j1 1j1 1j1 1j1 │
│1 1 1   1   1 1j1     │1 1j1 1j1 1j1 1j1 1j1 │
└──────────────────────┴──────────────────────┘
```

These results may be surprising; the functions do possess some of the characteristics to be expected, but they are clearly not defined as the floors and ceilings of the individual real and imaginary parts. This may be illustrated as follows:

```
   ri=: +.
   y=: 2r3j4r5
   (] ; ri ; j./@<.) y
┌─────────────┬───────────────┬───┐
│0.6666667j0.8│0.6666667 0.8  │0j1│
└─────────────┴───────────────┴───┘

   sfl=: j./@<.@+."0
   < sfl xm
┌──────────────┐
│0 0 0 0 0 0j1 │
│0 0 0 0 0 0j1 │
│0 0 0 0 0 0j1 │
│0 0 0 0 0 0j1 │
│0 0 0 0 0 0j1 │
│1 1 1 1 1 1j1 │
└──────────────┘
```

For a somewhat "higher-level" problem in the sense of GKP, one might experiment with `<.` and `>.` on further tables, and try to define the properties of a function that would lead to the definitions adopted for them in **J**. Any solution may be compared with the discussion in McDonnell [5] also cited in the dictionary of **J**.

Clues to the definition may be found by examining the region of the complex plane that maps to a particular Gaussian integer. For example:

```
   map=: {&'.*' @ (0j1&=) [. Si=: Ei@+: - ] [. Ei=: i.@>:
   (0j1= <.xm);(map <.xm);(0j1= >.xm);(map >.xm)
```

```
0 0 0 0 0 1 │.....*│0 1 1 1 1 1│.*****
0 0 0 0 1 1 │....**│0 1 1 1 1 0│.****.
0 0 0 1 1 1 │...***│0 0 1 1 0 0│..**..
0 0 0 0 1 1 │....**│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│......
```

```
   (map@<. ; map@>.) j./~ 1r5 * Si 10
```

```
.....................│.....................
.....................│.....................
.....................│.....................
.....................│.....................
.....................│.....................
.....................│.....................
.....................│...............*.....
.....................│..............***....
.................*...│.............*****...
................****.│............*****....
...............*****.│...........*****.....
..............*****..│...........****......
.............*****...│............**.......
..............***....│.....................
...............*.....│.....................
.....................│.....................
.....................│.....................
.....................│.....................
.....................│.....................
.....................│.....................
.....................│.....................
```

B. INTERVALS

The closed interval conventionally denoted by `a<x<b` may be defined by a proposition, to be used in the manner `x in a,b`. Thus:

```
   in=: >/@sgd
     sgd=: *@(-~/~)                     Sign of difference

   ]x=: 1r2*Si 5
_2.5 _2 _1.5 _1 _0.5 0 0.5 1 1.5 2 2.5

   x sgd 1 2
_1 _1 _1 _1 _1 _1 _1  0  1 1 1
_1 _1 _1 _1 _1 _1 _1 _1 _1 0 1

   x in 1 2
0 0 0 0 0 0 0 1 1 1 0

   x #~ x in 1 2
1 1.5 2
```

Similar definitions for the various closed and open intervals may be derived from the patterns observed in the result of `x sgd 1 2` above. Thus:

```
   incc=: >/@sgd                        Interval closed on left and right
   inco=: 0&>:@(%/)@sgd
   inoc=: e.&0 1@(+/)@sgd
   inoo=: 0&=@(+/)@sgd
   x ((incc#[);(inco#[);(inoc#[);(inoo#[)) 1 2
```

1 1.5 2	1 1.5	1.5 2	1.5

These functions may be combined in a gerund to define an adverb `IN` such that `0á0áIN` through `1 1 IN` (or, perhaps `0 IN` through `3 IN`) provide all cases.

C. RESIDUE

The sine function is periodic in the sense that it repeats after a certain period p; that is, sine (x+p) equals sine x for any x. We might therefore say that it is *congruent* with respect to the *measure* p, or *congruent modulo* p.

If the study of the sine had begun with emphasis on this important property it might well have been named modulo. Such a name would clearly be inappropriate since the sine is only one of many periodic functions, functions which include the cosine and the remainder or residue on division by p. Nevertheless, the term modulo (or mod) has gained wide acceptance for the latter function.

The function *mod* used in GKP is the commute of the residue denoted by `|` . Thus:

```
   3|i.8
0 1 2 0 1 2 0 1
   mod=: |~
   (i.8) mod 3
0 1 2 0 1 2 0 1
```

Using `c` to denote a *constant times* function, GKP3.23 can be expressed as a tautology as follows:

```
   c=: 5&*
   t1=: c@mod = mod&c
```

For example:

```
   7 t1 3
1
```

Similarly:

```
   t2=: c@| = |&c
```

The periodic properties of the residue (and a way of deriving a divisibility table from it by comparison with zero) may be seen in the following table:

```
    a=: Si 4
    (a By a Over a |/ a),&<(a By a Over 0 = a |/ a)
```
See §E

	_4	_3	_2	_1	0	1	2	3	4
_4	0	_3	_2	_1	0	_3	_2	_1	0
_3	_1	0	_2	_1	0	_2	_1	0	_2
_2	0	_1	0	_1	0	_1	0	_1	0
_1	0	0	0	0	0	0	0	0	0
0	_4	_3	_2	_1	0	1	2	3	4
1	0	0	0	0	0	0	0	0	0
2	0	1	0	1	0	1	0	1	0
3	2	0	1	2	0	1	2	0	1
4	0	1	2	3	0	1	2	3	0

	_4	_3	_2	_1	0	1	2	3	4
_4	1	0	0	0	1	0	0	0	1
_3	0	1	0	0	1	0	0	1	0
_2	1	0	1	0	1	0	1	0	1
_1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	1	0	1	0	1	0	1	0	1
3	0	1	0	0	1	0	0	1	0
4	1	0	0	0	1	0	0	0	1

Since the sum down a column of a divisibility table gives the number of distinct divisors, a simple test for primes may be defined as follows:

```
   prime=: 2: = +/@(0: = Ai | ])
     Ai=: >:@i.
```

```
   prime 5
1
   prime"0 a=: i. 20
0 0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1
   a #~ prime"0 a
2 3 5 7 11 13 17 19
```

The function `prime` may be compared with the proposition `IsPr` used in §2A.

The heart of the problem of partitioning `n` things into `m` groups as equally as possible (posed just after GKP3.23) is a function that yields the number in the first partition, the number remaining, and the number of groups decremented by one. Thus:

```
   f=: >.@% , ([ - >.@%) , (<:@])
   314 f 6
53 261 5
   f/ 314 6
53 261 5
```

Recursive use of `f` to append these results until the value of `m` reaches zero may be done as follows:

```
   g=: _2&}.`($:@(_2&}. , f/@(_2&{.)))@.(*@{:)
   g 314 6
53 53 52 52 52 52
```

D. FLOOR AND CEILING SUMS

§3.5 of GKP concerns the sum `+/@ipsqr@Ei`, where `ipsqr` is the integer part of the square root. Thus:

```
   sr=: +/@ipsqr@Ei
     ipsqr=: <.@%:
   sr"0 i. 10
0 1 2 3 5 7 9 11 13 16
```

Alternative functions for this sum given in GKP may be expressed using the interval functions defined in §B.

E. NOTATION

Bordering. The functions `By` and `Over` used in §C (and in other chapters) are defined by:

```
   By=: ' '&;@,.@[ ,. ]  [.  Over=: ({.;}.)@":@,
```

Compose and atop. For monadic use (as in `f&g y` and `f@g y`) these conjunctions are equivalent, but in dyadic use `x f&g y` is defined by `(g x) f (g y)`, whereas `x f@g y` is defined by `x f (g y)`.

Grade and sort. Used monadically, the function `/:` *grades* its argument; used dyadically, it permutes the left argument according to the grade of its right argument. For example:

```
   a=: 3 1 4 1 6
   b=: 'cable'
   (/:a) ; (b/:a) ; (a/:a) ; (/:b); (b/:b) ; (/:~a)
┌─────────┬─────┬─────────┬─────────┬─────┬─────────┐
│1 3 0 2 4│alcbe│1 1 3 4 6│1 2 0 4 3│abcel│1 1 3 4 6│
└─────────┴─────┴─────────┴─────────┴─────┴─────────┘
```

Membership. The *membership* function `e.` is so denoted because the corresponding function in math is denoted by the Greek epsilon.

Chapter 4
NUMBER THEORY

A. DIVISIBILITY

As remarked in GKP, m | t has been used in mathematics to assert that m divides t. Since `|` denotes residue in **J**, we define a divide test as `divides=: 0:=|`, and will illustrate its use together with a table adverb `Ta` (to produce a bordered table for easy reading) defined as follows:

```
   Ta=: / ([`By`]`Over`)\
     By=: ' '&;@,.@[ ,. ]  [.  Over=: ({.;}.)@":@,
   divides=: 0:=|
   divides"0 Ta
[ By ] Over divides"0/
```

```
   (Ai divides"0 Ta Ai) 6
```

The *augmented indices* `Ai=: >:@i.`

	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	0	1	0	1
3	0	0	1	0	0	1
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

As stated in GKP4.2 , the greatest common divisor may be defined by:

```
   gcd=: >./@cd
     cd=: Ai@<. #~ */"1 @ (Ai@<. divides"0/ ,)
   (Ai gcd"0 Ta Ai) 9
```

Greatest of common divisors

	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1
2	1	2	1	2	1	2	1	2	1
3	1	1	3	1	1	3	1	1	3
4	1	2	1	4	1	2	1	4	1
5	1	1	1	1	5	1	1	1	1
6	1	2	3	2	1	6	1	2	3
7	1	1	1	1	1	1	7	1	1
8	1	2	1	4	1	2	1	8	1
9	1	1	3	1	1	3	1	1	9

Components of the definition of `cd` (common divisors) may be illustrated as follows:

```
   dt=: Ai@<. divides"0/ ,
   ,.&.> 9(Ai@<. ; dt ; */"1@dt;(Ai@<. #~ */"1@dt))6
```

1	1 1	1	1
2	0 1	0	3
3	1 1	1	
4	0 0	0	
5	0 0	0	
6	0 1	0	

GKP4.2 is written as a function of a list (that is, as gcd (m,t)) rather than as a function of two arguments, and therefore corresponds to `gcd/` or `gcd/"1` rather than to `gcd` itself. A function for the least common multiple can be defined analogously according to GKP4.3, but we will use the somewhat more generally defined functions `+.` and `*.` :

```
   Si=: ]-Ei@+: [. Ei=: i.@>:
```

```
   ((Si +. Ta Si),.(Si *. Ta Si)) 4
```

	_4	_3	_2	_1	0	1	2	3	4		_4	_3	_2	_1	0	1	2	3	4
_4	4	1	2	1	4	1	2	1	4	_4	4	12	4	4	0	_4	_4	_12	_4
_3	1	3	1	1	3	1	1	3	1	_3	12	3	6	3	0	_3	_6	_3	_12
_2	2	1	2	1	2	1	2	1	2	_2	4	6	2	2	0	_2	_2	_6	_4
_1	1	1	1	1	1	1	1	1	1	_1	4	3	2	1	0	_1	_2	_3	_4
0	4	3	2	1	0	1	2	3	4	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	_4	_3	_2	_1	0	1	2	3	4
2	2	1	2	1	2	1	2	1	2	2	_4	_6	_2	_2	0	2	2	6	4
3	1	3	1	1	3	1	1	3	1	3	_12	_3	_6	_3	0	3	6	3	12
4	4	1	2	1	4	1	2	1	4	4	_4	_12	_4	_4	0	4	4	12	4

On the boolean sub-domain `0 1`, the functions `+.` and `*.` are *or* and *and*. Since `+.` and `*.` are associative and commutative, the functions `+./` and `*./` are symmetric:

```
   ranp=: ?~@#{]                        A random permutation
   x=: 96 24 48 36
   y=: ranp x
   x;y;(+./x);(+./y);(*./x);(*./y)
```

96 24 48 36	96 48 36 24	12	12	288	288

```
   T1=: +./ -: +./@ranp
   T2=: *./ -: *./@ranp
   T3=: *. = * % +.                     LCM is the product divided by the GCD
   gcdr=: gcdr@(|/ , {.)`{: @.(0:={.)          Recursive definition
   gcdr 24 60
12
   gcdr 60 24
12
```

The identity stated in GKP4.4 may be expressed as follows:

```
   GKP4_4=: +./ = +./@(|/ , {.)
   GKP4_4"1 ?20 2 $ 1000
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Consequently, the function `g=: |/ , {.` may be applied repeatedly to yield pairs that have the same GCD. For example:

```
   g=: |/ , {.
   g^:0 1 2 3 4 5 y=: 228 39
228  39
 39 228
 33  39
  6  33
  3   6
  0   3
   gcdp=: {.@(g^:(*@(|/))^:_)
   gcdp y
3
```

The foregoing function `gcdp` is defined to apply `g` until the remainder becomes zero, and to then select the first element of the result. The process may be modified to produce the integer quotient as well, and to record the successive quotients in the result:

```
   4!:55 'q';'r';'f'                    Erase names (undefined names are
1 1 1                                   treated as verbs until defined - See §1E)
   qrf=: q , r , f                      Quotient, remainder, first
     q=: f %~ {: - r=: |/ [. f=: {.
```

```
   qrf x=: 13 76
5 11 13
```

`   qrfI=: {: , 1&{ + {. * {:á` Inverse function

```
   qrfI qrf x
13 76
```

By applying `qrf` to the last two elements and appending its result to the remaining elements we obtain a process, called the Euclidean algorithm, that may be applied repeatedly. An analogous extension of the inverse function may also be made:

```
   eu=: _2&}. , qrf@(_2&{.)
   euI=: _3&}. , qrfI@(_3&{.)
   eu&.> ^: 0 1 2 3 4 5 <x
```

13 76	5 11 13	5 1 2 11	5 1 5 1 2	5 1 5 2 0 1	5 1 5 2 0 1 0

```
   euI&.> ^: 0 1 2 3 4 5 <eu ^:5 x
```

5 1 5 2 0 1 0	5 1 5 2 0 1	5 1 5 1 2	5 1 2 11	5 11 13	13 76

Although `eu` to any power continues to give a correct result (to which `euI` applies correctly), neither function terminates and they must be terminated by tests, using either a gerund and agenda or a power of the function under self-reference:

```
   euc=: $:@eu ^: (*@(|/@(_2&{.)))
   eucI=: $:@euI ^:(2:<#)
   ];euc;eucI@euc) x
```

13 76	5 1 5 1 2	13 76

B. THE EUCLIDEAN ALGORITHM

The process defined by `qrf` can be meaningful for a wide variety of the component functions `q` and `r`. They may, for example, concern the remainder and quotient on dividing one polynomial (represented by its coefficients) by another. We will sketch an approach to this as follows:

```
   rem=: 1: }. -/@(] ,: [ * %~&{.)
    m rem t [ m=: 2 3 4 [ t=: 6 4 2 1 7
_5 _10 1 7
   m rem m rem t
_2.5 11 7
```

Since the arguments are not scalars, we will re-express the process in terms of boxed arguments:

```
   brem=: {. , {. rem&.> {:
   (brem b) ,&< (brem brem b=: m ; t)
```

2 3 4	_5 _10 1 7	2 3 4	_2.5 11 7

C. NUMBER SYSTEMS

The expression `b #. d` yields the base `b` value of a list of digits `d`. For example:

```
   (10 #. 1 9 9 5);(2 #. 1 0 1 1);(8 #. 1 0 1 1)
```

1995	11	521

A function `f` that yields all numbers (or at least a significant subset such as all non-negative integers) will be said to define a *number system*, and if `n=: f d`, then `d` is said to *represent* `n` in the system defined by `f`. For example, `10&#.` defines the decimal system. Number systems have useful properties, such as those illustrated below:

```
   dec=: 10&#.
   a=: 3 6 5 [ b=: 3 1 4
   ((dec a)+(dec b)) ; (a+b) ; (dec a+b)
┌───┬─────┬───┐
│679│6 7 9│679│
└───┴─────┴───┘
   ((dec a)*(dec b));(a +//.@(*/) b);(dec a +//.@(*/) b)
┌──────┬─────────────┬──────┐
│114610│9 21 33 29 20│114610│
└──────┴─────────────┴──────┘
```

Since the product `9 21 33 29 20` could also be represented by `1 1 4 6 1 0`, it is clear that representations under `dec` are not unique, and that a difference in two representations does not imply that they represent different numbers. Uniqueness under `dec` can be ensured by restricting elements to non-negative integers less than `10`, and suppressing leading zeros.

The phrase `a+b` used above worked only because `a` and `b` had the same number of elements. More generally, the shorter of two such lists must be prefaced by zeros before adding. A similar problem arises in the addition of polynomial coefficients, where the function `ps` must append trailing zeros. The corresponding sum functions may be defined as follows:

```
   ps=: +/@,:
   ds=: ps&.|.
   3 6 5 (ps ,: ds) 3 1 4 1 5
6 7 9 1  5
3 1 7 7 10
```

A number system based upon prime numbers provides interesting expressions for the greatest common divisor and least common multiple, and expressions for multiplication and division that are analogous to logarithms. For example:

```
   a=: 2 0 2 0 1
   b=: 1 1 1 1 1
   ]pr=: p: i. # a              First #a primes
2 3 5 7 11
   pr^a                         Powers of primes
4 1 25 1 11
   */pr^a                       Number represented by a
1100
   f=: */@(pr&^)                A number system
   (f a);(f b);((f a)+.(f b));(a<.b);(f a<.b)             GCD
┌────┬────┬───┬─────────┬───┐
│1100│2310│110│1 0 1 0 1│110│
└────┴────┴───┴─────────┴───┘
   (f a);(f b);((f a)*.(f b));(a>.b);(f a>.b)             LCM
┌────┬────┬─────┬─────────┬─────┐
│1100│2310│23100│2 1 2 1 1│23100│
└────┴────┴─────┴─────────┴─────┘
   (f a);(f b);((f a)*(f b));(a+b);(f a+b)                Product
┌────┬────┬───────┬─────────┬───────┐
│1100│2310│2541000│3 1 3 1 2│2541000│
└────┴────┴───────┴─────────┴───────┘
```

```
   (f a);(f b);((f a)%(f b));(a-b);(f a-b)
```
Quotient

1100	2310	0.4761905	1 _1 1 _1 0	0.4761905

The function `f` must be re-defined to make it independent of the particular list of primes `pr`. Thus:

```
   f=: */@(p:@i.@# ^ ])
```

Moreover, expressions such as `a<.b` and `a+b` that occur in the foregoing examples will not work for lists that differ in number of elements, and must be replaced by expressions such as:

```
   plus=: +/@,: [. minus=: -/@,: [. max=:min&.- [. min=: <./@,:
   a=: 3 0 2
   b=: 1 1 1 1 1
   (f a);(f b);((f a)*.(f b));(a max b);(f a max b)
```
LCM

200	2310	46200	3 1 2 1 1	46200

```
   (f a);(f b);((f a)*(f b));(a plus b);(f a plus b)
```
Prod

200	2310	462000	4 1 3 1 1	462000

The inverse problem of determining the list of exponents that represent a number may be handled as follows:

```
   q: n=: 1100                        Factors
2 2 5 5 11
   pix=: p:^:_1@(_1&{.)               Prime index of last factor
   pix q: n
4
   primes=: p:@i.@>:@pix              Successive primes
   primes q: n
2 3 5 7 11
   (] =/ primes) q: n                 Classification of factors versus primes
1 0 0 0 0
1 0 0 0 0
0 0 1 0 0
0 0 1 0 0
0 0 0 0 1
   rep=: +/@(] =/ primes)@q:          Representation
   rep n
2 0 2 0 1
   f rep n
1100
```

D. FACTORIAL FACTORS

The Stirling approximation to the factorial (GKP4.23) may be expressed as:

```
   Saf=: %:@(2p1&*) * %&1x1 ^ ]
   (Saf , !,: Saf%!) i. 10
0 0.922137     1.919   5.83621   23.5062   118.019   710.078    4980.4   39902.4    359537
1        1         2         6        24       120       720      5040     40320    362880
0 0.922137 0.9595022 0.9727016 0.979424 0.9834931 0.9862197 0.9881738 0.9896427 0.9907872
   0.2 ": (! , Saf) 10
```

```
3628800.00 3598695.62
```

A straightforward function for the largest power of a prime `p` that divides `!n` is obtained by counting the occurrences of `p` in the factorization of `!n`:

```
   powin=: +/@([ = q:@!@])"0
   ]primes =: p: i. 6
2 3 5 7 11 13
   primes powin/ i.13
0 0 1 1 3 3 4 4 7 7 8 8 10
0 0 0 1 1 1 2 2 2 4 4 4  5
0 0 0 0 0 1 1 1 1 1 2 2  2
0 0 0 0 0 0 0 1 1 1 1 1  1
0 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
```

This function is, of course, limited to factorials that are precisely representable in the computer system used. A more usable function (defined by GKP4.25) may be expressed as follows:

```
   POWIN=: +/@<.@(] % [ ^ Ai@<.@(^. 1&>.))"0
     Ai=: >:@i.
   primes POWIN/ i.21
0 0 1 1 3 3 4 4 7 7 8 8 10 10 11 11 15 15 16 16 18
0 0 0 1 1 1 2 2 2 4 4 4  5  5  5  6  6  6  8  8  8
0 0 0 0 0 1 1 1 1 1 2 2  2  2  2  3  3  3  3  3  4
0 0 0 0 0 0 0 1 1 1 1 1  1  1  2  2  2  2  2  2  2
0 0 0 0 0 0 0 0 0 0 0 1  1  1  1  1  1  1  1  1  1
0 0 0 0 0 0 0 0 0 0 0 0  0  1  1  1  1  1  1  1  1
   53 POWIN 52 53 100 1000 10000
0 1 1 18 191
```

E. RELATIVE PRIMALITY

Just as a test for divisibility may be expressed as `0:=|`, so a test for relative primality as defined by GKP4.26 may be expressed as `1:=+.`. Thus:

```
   dt=: (0:=|)"0
   rp=: (1:=+.)"0
   ((Ai rp Ta Ai),.(Ai dt Ta Ai)) 10
┌──┬────────────────────┬──┬────────────────────┐
│  │1 2 3 4 5 6 7 8 9 10│  │1 2 3 4 5 6 7 8 9 10│
├──┼────────────────────┼──┼────────────────────┤
│ 1│1 1 1 1 1 1 1 1 1  1│ 1│1 1 1 1 1 1 1 1 1  1│
│ 2│1 0 1 0 1 0 1 0 1  0│ 2│0 1 0 1 0 1 0 1 0  1│
│ 3│1 1 0 1 1 0 1 1 0  1│ 3│0 0 1 0 0 1 0 0 1  0│
│ 4│1 0 1 0 1 0 1 0 1  0│ 4│0 0 0 1 0 0 0 1 0  0│
│ 5│1 1 1 1 0 1 1 1 1  0│ 5│0 0 0 0 1 0 0 0 0  1│
│ 6│1 0 0 0 1 0 1 0 0  0│ 6│0 0 0 0 0 1 0 0 0  0│
│ 7│1 1 1 1 1 1 0 1 1  1│ 7│0 0 0 0 0 0 1 0 0  0│
│ 8│1 0 1 0 1 0 1 0 1  0│ 8│0 0 0 0 0 0 0 1 0  0│
│ 9│1 1 0 1 1 0 1 1 0  1│ 9│0 0 0 0 0 0 0 0 1  0│
│10│1 0 1 0 0 0 1 0 1  0│10│0 0 0 0 0 0 0 0 0  1│
└──┴────────────────────┴──┴────────────────────┘
```

If `a` is a two-element list of integers that are relatively prime (that is, `rp/a` is true), then `a` is said to be the representation of the fraction `%/a` in lowest form. Moreover, `b % +./b` is necessarily in lowest form.

The function `sb` defined below expands its list argument by inserting the sum of each adjacent pair between them. For example:

```
   pair=: 1 :'2: x.\ ]'          Applies function argument over pairs
   a=: 3 1 4 1 5 9
   +/ pair a
4 5 5 6 14
```

```
   sb=: {: ,~ [: , +/\pair              Verb after cap ([:) applies monadically
   sb a
3 4 1 5 4 5 1 6 5 14 9
```

As stated in GKP, the repeated application of `sb` to the arguments `0 1` and `1 0` generates the list of numerators and denominators of all fractions in lowest form. Moreover, the fractions they represent are in ascending order. For example:

```
   m=: sb^:4 (0 1) [ t=: sb^:4 (1 0)
   m,:t
0 1 1 2 1 3 2 3 1 4 3 5 2 5 3 4 1
1 4 3 5 2 5 3 4 1 3 2 3 1 2 1 1 0
```

The functions `rp` and `Mi=: ]-:/:~` may be used to show that `m` and `t` are relatively prime and that the fractions they represent are monotone increasing:

```
   (m rp t) ; (Mi m%t)
┌─────────────────────────────────┬─┐
│1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1│1│
└─────────────────────────────────┴─┘
```

GKP4.31 may be expressed as the negative of the determinant of the matrices represented by successive pairs of rows of the matrix `m ,. t`. Thus:

```
   -@Det pair m,.t [. Det=: -/ . *
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The following function produces the Stern-Brocot numerators and denominators of specified orders:

```
   sbr=: 3 : 'sb"1^:y. 0 1,:1 0'
   sbr 3
0 1 1 2 1 3 2 3 1
1 3 2 3 1 2 1 1 0

   sbr 2 3 4 5
0 1 1 2 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
1 2 1 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 1 1 2 1 3 2 3 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
1 3 2 3 1 2 1 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 1 1 2 1 3 2 3 1 4 3 5 2 5 3 4 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 4 3 5 2 5 3 4 1 3 2 3 1 2 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 2 1 3 2 3 1 4 3 5 2 5 3 4 1 5 4 7 3 8 5 7 2 7 5 8 3 7 4 5 1
1 5 4 7 3 8 5 7 2 7 5 8 3 7 4 5 1 4 3 5 2 5 3 4 1 3 2 3 1 2 1 1 0
```

Farey series are those members of `sbr` in which the numerator does not exceed the denominator, and the denominator does not exceed the order. Thus:

```
   farey=: ] (] #~"1 <:/@] *. [ >: {:@]) sbr
   farey 6
0 1 1 1 1 2 1 3 2 3 4 5 1
1 6 5 4 3 5 2 5 3 4 5 6 1
```

As stated in GKP, the test `-@Det pair |: farey 6` may also be applied to the Farey series to yield results of `1`.

"When N is prime, N-1 new fractions will appear; but otherwise we'll have fewer than N-1 because this process generates only numerators that are relatively prime to N.". This assertion in GKP may be compared with the following result for the number of elements in Farey series of various orders:

```
   ]y=: {:@$@farey"0 i.13
0 2 3 5 7 11 13 19 23 29 33 43 47
```

Because the list `sbr t` contains all members of the Stern-Brocot integers of order `t`, they cannot be compared directly with the tree display of S-B numbers provided in GKP. However, the selection of only those in alternate positions provides a result that may be so compared. Moreover, rows of the table may be shifted and formatted for more direct comparison. Thus:

```
   sbt=: (alt=: (2&|@i.@# # ])"1) sbr 0 1 2 3 4 5
   i=: |.+/\0 4 2 1 0 0
   sbsh=: (-i,.i) |."0 1 sbt
```

Numeric tables, such as those we have used thus far, can be formatted (using `":`), and the irrelevant zeros replaced by spaces to provide a less cluttered display. A simpler rough formatting can be provided by simple indexing. It works well in cases where the table to be displayed has no significant zeroes (such as those that occur in multi-digit numbers), as illustrated below.

```
   sbt;sbsh
┌───────────────────────────────┬───────────────────────────────┐
│1 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│
│                               │                               │
│1 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│
│1 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│
│                               │                               │
│1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0│0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0│
│2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0│0 0 0 0 0 0 0 2 1 0 0 0 0 0 0 0│
│                               │                               │
│1 2 3 3 0 0 0 0 0 0 0 0 0 0 0 0│0 0 0 0 0 0 1 2 3 3 0 0 0 0 0 0│
│3 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0│0 0 0 0 0 0 3 3 2 1 0 0 0 0 0 0│
│                               │                               │
│1 2 3 3 4 5 5 4 0 0 0 0 0 0 0 0│0 0 0 0 1 2 3 3 4 5 5 4 0 0 0 0│
│4 5 5 4 3 3 2 1 0 0 0 0 0 0 0 0│0 0 0 0 4 5 5 4 3 3 2 1 0 0 0 0│
│                               │                               │
│1 2 3 3 4 5 5 4 5 7 8 7 7 8 7 5│1 2 3 3 4 5 5 4 5 7 8 7 7 8 7 5│
│5 7 8 7 7 8 7 5 4 5 5 4 3 3 2 1│5 7 8 7 7 8 7 5 4 5 5 4 3 3 2 1│
└───────────────────────────────┴───────────────────────────────┘
```

We will use `sbsh` to index the character list `' 123456789'` producing a tree that may be compared with that on page 117 of GKP:

```
    sbsh { ' 123456789'
       1


       1
       1

       12
       21

      1233
      3321

    12334554
    45543321

1233455457877875
5787787545543321
```

The last item of `sbt` may be selected and then indexed to select a particular numerator/denominator pair. For example:

```
   ]q=: _1 { sbt
1 2 3 3 4 5 5 4 5 7 8 7 7 8 7 5
5 7 8 7 7 8 7 5 4 5 5 4 3 3 2 1
   6 {"1 q
5 7
```

If the index 6 is represented as the binary number `b=: 0 1 1 0` (that is, LRRL in the terminology of GKP), then the selection `(#.b){"1 q` may be construed as the selection of a path in the "binary tree" represented by `sbt`:

```
   #. b=: 0 1 1 0
6
   (path=: #. {"1 alt@sbr@>:@#) b
5 7
```

F. PHI and MU

Euler's *totient* or *phi* function of `m` is the number of elements of the list `i.m` that are relatively prime to `m`. Thus:

```
   phi=: +/@(i. rp ])"0
     rp=: (1: = +.)"0                    Test for relative primality
   (],:phi) i. 21
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
0 1 1 2 2 4 2 6 4 6  4 10  4 12  6  8  8 16  6 18  8
```

Fermat's Little theorem as stated in GKP4.47 may be expressed as follows:

```
   GKP4_47=: ([ rp p:@]) <: p:@] | [ ^ <:@p:@]
   p: y=: i. 5
2 3 5 7 11
   y GKP4_47"0/ y
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
```

Its generalization as stated in GKP4.50 may be expressed as:

```
   GKP4_50=: rp <: 1: = ] | [ ^ phi@]
   y GKP4_50"0/ y
1 0 1 1 1
1 0 1 1 1
1 0 1 1 1
1 0 1 1 1
1 0 1 1 1
```

The column of zeros indicates that the theorem fails for a left argument of `1` (m=1 in GKP). Since the residue of any integer modulo `1` is zero, the case m=1 should be excluded as : `GKP4_50=: (rp *. ] ~: 1:) <: ] | [ ^ phi@]`

A simple argument (given in GKP) shows that the number of divisors of the `k`th power of a prime `p` is simply the difference `(p^k)-(p^k-1)`. Hence the following theorem:

```
   L=: phi@(p:@] ^ [)
   R=: (p:@] ^ [) - (p:@] ^ [ - 1:)
   T=: L -: R
   1 2 3 (L"0/ ; R"0/ ; T"0/) 0 1 2 3 4
┌─────────────────┬─────────────────┬─────────┐
│1  2   4   6   10│1  2   4   6   10│1 1 1 1 1│
│2  6  20  42  110│2  6  20  42  110│1 1 1 1 1│
│4 18 100 294 1210│4 18 100 294 1210│1 1 1 1 1│
└─────────────────┴─────────────────┴─────────┘
```

If two integers are relatively prime they share no divisors, and a simple argument shows that the number of divisors of their product is the product of the number of divisors of each. For example:

```
   a=: 5^3 [ b=: 2^4
   a([ ; ] ; phi@[ ; phi@] ; phi@* ; */&phi ; rp) b
┌───┬──┬───┬─┬───┬───┬─┐
│125│16│100│8│800│800│1│
└───┴──┴───┴─┴───┴───┴─┘
```

An integer can be expressed as the product `*/p^e`, where `p` is a list of distinct primes, and `e` is a list of integer exponents. The list of prime factors produced by `q:` provides the basis for these as follows:

```
   dpr=: ~.@q:                    Distinct primes among prime factors
   pex=: #/.~@q:                  Number in each group of factors
   pde=: dpr ,: pex               Prime decomposition (primes and exponents)
   (q: ; dpr ; pex ; pde ; ^/@pde ; */@(^/)@pde) 1400
┌───────────┬─────┬─────┬─────┬──────┬────┐
│2 2 2 5 5 7│2 5 7│3 2 1│2 5 7│8 25 7│1400│
│           │     │     │3 2 1│      │    │
└───────────┴─────┴─────┴─────┴──────┴────┘
```

These results can now be used in an alternative definition of the function for the number of divisors:

```
   phi1=: */@(f/@pde)             Alternative totient function
     f=: ^ - [ ^ ] - 1:           From R (# of divisors in power of a prime)
   (phi1 ; phi ; pde ; f/@pde) 1400
┌───┬───┬─────┬───────┐
│480│480│2 5 7│4 20 6 │
│   │   │3 2 1│       │
└───┴───┴─────┴───────┘
```

The execution of `phi1` is, of course, much faster than `phi`. Timings can be obtained as follows:

```
   time=: 6!:2
   100 time 'phi1 490'            Average time for 100 executions
0.0368
   100 time 'phi 490'
0.2686
```

The right limb of GKP4.53 can be used to define a further totient function as follows:

```
   phi2=: [ * -.@%@~.&.q:
   100 time 'phi2 490'
0.0126
```

The following functions provide a list of *basic rationals* (whose quotients yield a list of *basic fractions* in the range from zero to nearly one), *reduction* to a corresponding table with relatively prime numerators and denominators, and *sorting* on the denominators:

```
   S=: srt@red@bar"0
     red=: ] %"1 +./ [. bar=: i.,:]
     srt=: /:"1 {:
   ,.(S ; red@bar ; bar) 12
┌───────────────────────────────────┐
│0 1 1 2 1 3 1 5  1  5  7 11        │
│1 2 3 3 4 4 6 6 12 12 12 12        │
├───────────────────────────────────┤
│0  1 1 1 1  5 1  7 2 3 5 11        │
│1 12 6 4 3 12 2 12 3 4 6 12        │
├───────────────────────────────────┤
│ 0  1  2  3  4  5  6  7  8  9 10 11│
│12 12 12 12 12 12 12 12 12 12 12 12│
└───────────────────────────────────┘
```

As remarked in GKP, every divisor `d` of `12` occurs as a denominator, together with all `phi d` of its numerators, so the sum over `phi` on all divisors must equal `12`. Thus:

```
   div=: Ei #~ 0: = Ei | ] [. Ei=: i.@>:
   (div ; phi@div ; +/@(phi@div))"0 (12 20 30 40)
│1 2 3 4 6 12         │1 1 2 2 2 4       │12│
```

1 2 4 5 10 20	1 1 2 4 4 8	20
1 2 3 5 6 10 15 30	1 1 2 4 2 4 8 8	30
1 2 4 5 8 10 20 40	1 1 2 4 4 4 8 16	40

The theorem of GKP4.54 may therefore be expressed as follows:

```
   GKP4_54=: ] -: +/@(phi@div)
```

The definition of the Mobius function in GKP4.57 may be expressed in terms of the prime exponents as follows:

```
   mu=: */@(_1&^@# , ] = 1:)@ pex
   (mu"0 ,: ]) >: i. 20
1 _1 _1 0 _1 1 _1 0 0  1 _1  0 _1  1  1  0 _1  0 _1  0
1  2  3 4  5 6  7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Other properties of `mu` and `phi` presented in GKP theorems may be expressed rather simply in terms of them (or in terms of `phi1` if speed of execution is important). For example:

```
   L=: phi
   R=: +/@( mu@div * ] % div)"0
   GKP4_58=: L -: R
   (L , R,: GKP4_58) 1 + i. 20
1 1 2 2 4 2 6 4 6 4 10 4 12 6 8 8 16 6 18 8
1 1 2 2 4 2 6 4 6 4 10 4 12 6 8 8 16 6 18 8
1 1 1 1 1 1 1 1 1 1  1 1  1 1 1 1  1 1  1 1
```

G. NOTATION

Columnize under. The function `,.` *columnizes* a vector; used with *under open* (`&.>`) it columnizes boxed vectors. For example:

```
   a=: 1 2 3 [. b=: 4 5 6 7
   a;(,.a);a+/b
┌─────┬─┬────────┐
│1 2 3│1│5 6 7  8│
│     │2│6 7 8  9│
│     │3│7 8 9 10│
└─────┴─┴────────┘
   ,.&.> a;b;a+/b
┌─┬─┬────────┐
│1│4│5 6 7  8│
│2│5│6 7 8  9│
│3│6│7 8 9 10│
│ │7│        │
└─┴─┴────────┘
```

Grade and sort. `/: y` grades the argument `y`, and `x /: y` permutes `x` according to the grade of `y`. Downgrade is denoted by `\:` .

Nub. The function `~.` suppresses all repeated items from its argument. For example:

```
   ~. 'mississippi'
misp
```

Explicit definition. The sentence `pair=: 1 :'2: x.\ ]'`that defines the adverb `pair` is an example of explicit definition, in which `x.` refers to the left argument of the adverb.

Transpose. The function `|:` reverses the order of the axes of its argument. For example:

```
   a=: i. 3 4 [ b=: i. 2 3 4
   a ; (|:a) ; b ; (|:b)
```

```
│0 1  2  3│0 4  8│ 0  1  2  3│ 0 12│
│4 5  6  7│1 5  9│ 4  5  6  7│ 4 16│
│8 9 10 11│2 6 10│ 8  9 10 11│ 8 20│
│         │3 7 11│           │     │
│         │      │12 13 14 15│ 1 13│
│         │      │16 17 18 19│ 5 17│
│         │      │20 21 22 23│ 9 21│
│         │      │           │     │
│         │      │           │ 2 14│
│         │      │           │ 6 18│
│         │      │           │10 22│
│         │      │           │     │
│         │      │           │ 3 15│
│         │      │           │ 7 19│
│         │      │           │11 23│
└─────────┴──────┴───────────┴─────┘
```

Laminate. The verb `,:` laminates its arguments to produce a result of higher rank, first padding a possibly shorter one to bring them to a common shape. For example:

```
   3 2 4 ,: 2 7 1 8 2
3 2 4 0 0
2 7 1 8 2
```

GCD, LCM. `+.` and `*.` yield the GCD and LCM which, for boolean arguments `0` and `1`, are equivalent to *or* and *and*.

Cap. When a cap (`[:`) occurs in a fork, it acts as a "null", causing the verb that follows it to apply monadically. For example, the comma in the following definition acts to ravel its right argument rather than to catenate it to anything:

```
   sb=: {: ,~ [: , +/\pair
```

Chapter 5
BINOMIAL COEFFICIENTS

A. BASIC IDENTITIES

The binomial theorem expresses a power of a sum (that is, `(x+y)^t`) as an equivalent weighted sum of products of ascending powers of `x` and descending powers of `y`:

```
   x=: 3 [ y=: 5 [ t=: 4 [. Ei=: i.@>:
   (x+y)^t
4096
   (Ei  t);(x^Ei t);(|.Ei t);(y^|.Ei t)
```

0 1 2 3 4	1 3 9 27 81	4 3 2 1 0	625 125 25 5 1

```
   +/ 1 4 6 4 1 * (x^Ei t) * (y^|.Ei t)
4096
```

The weights (`1 4 6 4 1` in the case of `t=: 4`) are called the *binomial coefficients of order* `t`. It remains to determine a function `bc` that yields the binomial coefficients of the order of its argument.

For the case `y=: 1` all powers of `y` are `1`, and the identity reduces to the form:

```
   ((x+1)^t) = +/(bc t) * x ^ Ei t
```

Since `(x+1)^ t` is a product of `t` factors `x+1`, we may begin to determine the values of `bc t` by multiplication as follows:

```
                                 x+1
                                 x+1
                                 x+1
                             (x^2)+x
                        (x^2)+x+x+1
                                 x+1
                        (x^2)+x+x+1
                (x^3)+(x^2)+(x^2)+x
(x^3)+(x^2)+(x^2)+(x^2)+x+x+x+1
```

or

```
(1*x^3)+(3*x^2)+(3*x^1)+(1*x^0)
```

A term `x^s` occurs each time that `x` is chosen from exactly `s` of the `t` factors `x+1`, and the weight (i.e., binomial coefficient) to be assigned to `x^s` is therefore the number of ways that `s` things may be chosen from `t` things. This may be illustrated using the complete classification table `#: i. 2^t`:

```
   x=: 10 [ t=: 3
   cct=: #: i. 2^t
   cct;(x^cct);(,. p);(+/,p=: */"1 x^cct)
```

0 0 0	1 1 1	1	1331
0 0 1	1 1 10	10	
0 1 0	1 10 1	10	
0 1 1	1 10 10	100	
1 0 0	10 1 1	10	
1 0 1	10 1 10	100	
1 1 0	10 10 1	100	
1 1 1	10 10 10	1000	

```
   (x+1)^t
1331
```

In choosing `s` elements from `t`, the first may be chosen in `t` ways, and the next in `t-1` ways, and so on. Therefore, the number of choices for positions are `t-i.s`, and the product `*/t-i.s` gives the number of possible sequences. Since each of the `!s` permutations of any set of elements occurs among the sequences, the number of distinct selections is obtained by dividing `*/t-i.s` by `!s`. For example:

```
   ]t-i.s=: 3 [ t=: 5
5 4 3
   (*/t-i.s) % !s
10
```

In discussing binomial coefficients we will therefore make use of a dyadic case (subsiding integers) of the function `Si=: (Ei@+: - ]) : (-/i.)`. Thus:

```
   (8 Si 5) ; (8 5 Si 5) ; (5 Si 5)          Subsiding integers
┌─────────┬─────────┬─────────┐
│8 7 6 5 4│8 7 6 5 4│5 4 3 2 1│
│         │5 4 3 2 1│         │
└─────────┴─────────┴─────────┘
```

The factorial function `!` is the product over the sequence `5 Si 5`, and the number of combinations of `t` things chosen `s` at a time (denoted by `s!t`) is the quotient of the products over the rows of `(s,t) Si t`. For example:

```
   t=: 8 5 Si 5
   t ; (*/"1 t) ; (%/*/"1 t) ; (5 ! 8) ; (Ei !/ Ei) 5
┌─────────┬────────┬──┬──┬────────────┐
│8 7 6 5 4│6720 120│56│56│1 1 1 1 1  1│
│5 4 3 2 1│        │  │  │0 1 2 3 4  5│
│         │        │  │  │0 0 1 3 6 10│
│         │        │  │  │0 0 0 1 4 10│
│         │        │  │  │0 0 0 0 1  5│
│         │        │  │  │0 0 0 0 0  1│
└─────────┴────────┴──┴──┴────────────┘
```

The appearance of this table differs markedly from Pascal's triangle as shown in Table 155 of GKP, and it will be important to understand the relation of `!` to the corresponding function defined by GKP5.1. We will first state the major differences, and then examine the consequences, including consequences for a number of the theorems of GKP. Thus:

a) A polynomial coefficient may be extended by zeros without changing its significance, and such extension of the first `n` binomial coefficients yields a square matrix such as that shown above. Such a matrix can be used in significant ways; for example, the inverse yields the *alternating* binomial coefficients, as shown below.

b) The table `(Ei !/ Ei) 10` is the *transpose* of Table 155 of GKP, which would be given by the "passive" `choose=: !~`. The function `!` comes from the definition in GKP5.1 by interpreting the **L**ower index as the **L**eft argument, and `choose` comes from the opposite choice.

c) The extension of `!` to the case of two negative arguments (based on the gamma function) gives non-zero values where the extension in GKP gives zeros.

The binomial coefficients function `Bc` is defined by `Bc=: i. !/ i.`. The table it produces exhibits interesting properties under matrix inverse and the matrix product `X=: +/ . *`. For example:

```
   (%.m) ; m ; (m X m) ; (m X m X m=: Bc 5)
┌────────────┬─────────┬────────────┬─────────────┐
│1 ¯1  1 ¯1 1│1 1 1 1 1│1 2 4  8 16│1 3 9 27  81│
│0  1 ¯2  3 ¯4│0 1 2 3 4│0 1 4 12 32│0 1 6 27 108│
│0  0  1 ¯3  6│0 0 1 3 6│0 0 1  6 24│0 0 1  9  54│
│0  0  0  1 ¯4│0 0 0 1 4│0 0 0  1  8│0 0 0  1  12│
│0  0  0  0  1│0 0 0 0 1│0 0 0  0  1│0 0 0  0   1│
└────────────┴─────────┴────────────┴─────────────┘
```

The function `Epa=: Bc@# X ]` is pre-multiplication by the matrix `m`. Applied to a vector of coefficients it expands it to give the coefficients of a polynomial equivalent to `c p. x+1`. For example:

```
   c ; Epa c=: 3 1 4 2 1
┌─────────┬────────────┐
│3 1 4 2 1│11 19 16 6 1│
└─────────┴────────────┘
   (c p. x+1) ; (Epa c) p. x=: i. 6
┌──────────────────────┬──────────────────────┐
│11 53 177 455 983 1881│11 53 177 455 983 1881│
└──────────────────────┴──────────────────────┘
   (c p. x+2) ; (Epa Epa c) p. x=: i. 6
┌─────────────────────────┬─────────────────────────┐
│53 177 455 983 1881 3293│53 177 455 983 1881 3293│
└─────────────────────────┴─────────────────────────┘
```

The reason is that the columns of the matrix of coefficients are the expansions of successive powers, and the product `m X c` is the sum of the columns weighted by the coefficients `c`. The successive powers of `m` itself are equally interesting, and display a pattern that is most easily discerned by dividing (using ordinary element-by-element division) the powers by `m` itself:

```
   (<"2 (Epa^:1 2 3 m) %"2 m),<((4:^-~/~) * (<:/~)) i.#m
┌──────────┬───────────┬──────────────┬──────────────┐
│1 2 4 8 16│1 3 9 27 81│1 4 16 64 256│1 4 16 64 256│
│0 1 2 4  8│0 1 3  9 27│0 1  4 16  64│0 1  4 16  64│
│0 0 1 2  4│0 0 1  3  9│0 0  1  4  16│0 0  1  4  16│
│0 0 0 1  2│0 0 0  1  3│0 0  0  1   4│0 0  0  1   4│
│0 0 0 0  1│0 0 0  0  1│0 0  0  0   1│0 0  0  0   1│
└──────────┴───────────┴──────────────┴──────────────┘
```

The dyadic case of `!` may be defined in terms of the monadic case as follows:

```
   outof=: (!@]%(!@(]-[))*!@[)"0
   (4 outof 9) , (4 ! 9)
126 126
```

The reason can be seen in the following pattern:

```
   (9 Si 4) ; ((9-4) Si (9-4)) ; (4 Si 4)
┌───────┬─────────┬───────┐
│9 8 7 6│5 4 3 2 1│4 3 2 1│
└───────┴─────────┴───────┘
```

Since `4 outof 9` is the product over the first box divided by that over the last, it is also the product over the first two (`!9`) divided by the product of the products over the last two, that is `(!9-4)*(!4)`.

This definition in terms of factorials provides a basis for a generalization of the dyad `!` to negative and non-integer arguments. First, the factorial is so generalized by basing it on the gamma function. For example:

```
   ]j=: 0.5 * i.9
0 0.5 1 1.5 2 2.5 3 3.5 4

   !j                              Agrees with factorial on integers
1 0.886227 1 1.32934 2 3.32335 6 11.6317 24

   !j+1
1 1.32934 2 3.32335 6 11.6317 24 52.3428 120

   (j+1)*!j                        as well as in this behaviour
1 1.32934 2 3.32335 6 11.6317 24 52.3428 120

   ! -j                            with alternating infinities at negative integers
1 1.77245 _ _3.54491 __ 2.36327 _ _0.945309 __
```

When used with negative left arguments, the function `!` may involve infinite values, but if one occurs in the numerator, one will also occur in the denominator. The dyad `!` is defined in [2] to exploit this fact as follows:

> For non-negative arguments `x!y` is the number of ways that `x` things can be chosen out of `y`. More generally, `(x!y)` is `(!y)%(!x)*(!y-x)` with the

understanding that infinities (occasioned by `!` on a negative integer) cancel if they occur in both numerator and denominator.

We will now display the function table of `!`:

```
   Ta=: / ([`By`]`Over`)\
     Over=: ({.;}.)@":@,
       By=: ' '&;@,.@[ ,. ]
   (Si !Ta Si) 5
┌──┬──────────────────────────────────────┐
│  │  _5  _4  _3 _2 _1 0 1 2 3 4  5       │
├──┼──────────────────────────────────────┤
│_5│   1  _4   6 _4  1 0 0 0 0 0  0       │
│_4│   0   1  _3  3 _1 0 0 0 0 0  0       │
│_3│   0   0   1 _2  1 0 0 0 0 0  0       │
│_2│   0   0   0  1 _1 0 0 0 0 0  0       │
│_1│   0   0   0  0  1 0 0 0 0 0  0       │
│ 0│   1   1   1  1  1 1 1 1 1 1  1       │
│ 1│  _5  _4  _3 _2 _1 0 1 2 3 4  5       │
│ 2│  15  10   6  3  1 0 0 1 3 6 10       │
│ 3│ _35 _20 _10 _4 _1 0 0 0 1 4 10       │
│ 4│  70  35  15  5  1 0 0 0 0 1  5       │
│ 5│_126 _56 _21 _6 _1 0 0 0 0 0  1       │
└──┴──────────────────────────────────────┘
```

GKP5.1 extends the binomial coefficients similarly, except that the case of negative left arguments (the top five rows in the foregoing table) are defined to be zero. The definitions therefore differ only in the case where both arguments are negative. Thus:

```
   GKP5_1=: (! * [ >: 0:)"0
   GKP5_1 /~ Si 5
   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  0 0 0 0 0 0  0
   1   1   1  1  1 1 1 1 1 1  1
  _5  _4  _3 _2 _1 0 1 2 3 4  5
  15  10   6  3  1 0 0 1 3 6 10
 _35 _20 _10 _4 _1 0 0 0 1 4 10
  70  35  15  5  1 0 0 0 0 1  5
_126 _56 _21 _6 _1 0 0 0 0 0  1
```

Since GKP tables that show Pascal’s triangle and its extension are transposes of those produced by `!`, the relation is best illustrated by the following expression:

```
   |:(Ei !/ Si) 5
1 _5 15 _35 70 _126
1 _4 10 _20 35  _56
1 _3  6 _10 15  _21
1 _2  3  _4  5   _6
1 _1  1  _1  1   _1
1  0  0   0  0    0
1  1  0   0  0    0
1  2  1   0  0    0
1  3  3   1  0    0
1  4  6   4  1    0
1  5 10  10  5    1
```

We will now explore some of the identities of GKP, denoting the left and right limbs by L and R, followed by digits to indicate the particular equation. However, we will base the exploration mainly on the function `!` of **J**, and will therefore expect some deviation from the identities presented in GKP. Moreover, we will define a table adverb `T` to make it more convenient to express the tables produced by various functions, as illustrated by the expression for the final panel in the following example:

```
   T=: ("0)/~
```

```
   ! T
!"0/~
   ]i=: Si 3
_3 _2 _1 0 1 2 3

   L5_4=: !
   R5_4=: (-~ ! ])"0
   (i L5_4/ i);(i R5_4/ i);(R5_4 T i)
┌───────────────────┬───────────────────┬───────────────────┐
│  1 _2  1 0 0 0 0  │  1 _2  1 0 0 0 0  │  1 _2  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  0  1 0 0 0 0  │  0  0  1 0 0 0 0  │  0  0  1 0 0 0 0  │
│  1  1  1 1 1 1 1  │  1  1  1 1 1 1 1  │  1  1  1 1 1 1 1  │
│ _3 _2 _1 0 1 2 3  │ _3 _2 _1 0 1 2 3  │ _3 _2 _1 0 1 2 3  │
│  6  3  1 0 0 1 3  │  6  3  1 0 0 1 3  │  6  3  1 0 0 1 3  │
│_10 _4 _1 0 0 0 1  │_10 _4 _1 0 0 0 1  │_10 _4 _1 0 0 0 1  │
└───────────────────┴───────────────────┴───────────────────┘
```

The use of integers ranging from `_3` to `3` illustrates that the definition of `!` allows the removal of the restriction n>:0 included in GKP5.4. Because `!` is defined in terms of the gamma function, the restriction to integers can also be removed. For example:

```
   ,. (L5_4 T i+0.5) ; >./|,(L5_4 T i+0.5)-(R5_4 T i+0.5)
┌─────────────────────────────────────────────────────────────┐
│1 _1.5 0.375 0.0625 0.0234375 0.01171875 0.006835938│
│0    1  _0.5 _0.125  _0.0625 _0.0390625 _0.02734375│
│0    0     1    0.5    0.375     0.3125   0.2734375│
│0    0     0      1      1.5      1.875      2.1875│
│0    0     0      0        1        2.5       4.375│
│0    0     0      0        0          1         3.5│
│0    0     0      0        0          0           1│
├─────────────────────────────────────────────────────────────┤
│8.67362e_19                                                  │
└─────────────────────────────────────────────────────────────┘
```

The restriction to non-zero `s` in GKP5.5 can also be removed, due in part to the fact that `0%0` is defined as `0` in J. Thus:

```
   L5_5=: !
   R5_5=: (%~ * !&<:)
   (L5_5 T i);(R5_5 T i);(L5_5 T i)=(R5_5 T i)
┌───────────────────┬───────────────────┬─────────────┐
│  1 _2  1 0 0 0 0  │  1 _2  1 0 0 0 0  │1 1 1 1 1 1 1│
│  0  1 _1 0 0 0 0  │  0  1 _1 0 0 0 0  │1 1 1 1 1 1 1│
│  0  0  1 0 0 0 0  │  0  0  1 0 0 0 0  │1 1 1 1 1 1 1│
│  1  1  1 1 1 1 1  │  0  0  0 0 0 0 0  │0 0 0 0 0 0 0│
│ _3 _2 _1 0 1 2 3  │ _3 _2 _1 0 1 2 3  │1 1 1 1 1 1 1│
│  6  3  1 0 0 1 3  │  6  3  1 0 0 1 3  │1 1 1 1 1 1 1│
│_10 _4 _1 0 0 0 1  │_10 _4 _1 0 0 0 1  │1 1 1 1 1 1 1│
└───────────────────┴───────────────────┴─────────────┘
```

As shown by the row of zeros in the last panel, the identity fails in the case of a zero left argument (a case explicitly excluded in GKP5.5). In the case of GKP5.8 we find a discrepancy at the mid-point, that is, for arguments `0 0`, but can (as expected) remove it by using `GKP5_1` instead of `!`:

```
   L5_8=: ! [. R5_8=: ([ ! <:@])+(!&<:)
   (L5_8 T i);(R5_8 T i);(L5_8 T i)=(R5_8 T i)
┌───────────────────┬───────────────────┬─────────────┐
│  1 _2  1 0 0 0 0  │  1 _2  1 0 0 0 0  │1 1 1 1 1 1 1│
│  0  1 _1 0 0 0 0  │  0  1 _1 0 0 0 0  │1 1 1 1 1 1 1│
│  0  0  1 0 0 0 0  │  0  0  1 0 0 0 0  │1 1 1 1 1 1 1│
│  1  1  1 1 1 1 1  │  1  1  1 2 1 1 1  │1 1 1 0 1 1 1│
│ _3 _2 _1 0 1 2 3  │ _3 _2 _1 0 1 2 3  │1 1 1 1 1 1 1│
│  6  3  1 0 0 1 3  │  6  3  1 0 0 1 3  │1 1 1 1 1 1 1│
│_10 _4 _1 0 0 0 1  │_10 _4 _1 0 0 0 1  │1 1 1 1 1 1 1│
```

```
   L5_8a=: GKP5_1
   R5_8a=:([ GKP5_1 <:@])+(GKP5_1&<:)
   (L5_8a T i);(R5_8a T i);(L5_8a T i)=(R5_8a T i)
┌─────────────────────┬─────────────────────┬─────────────┐
│  0  0  0 0 0 0 0    │  0  0  0 0 0 0 0    │1 1 1 1 1 1 1│
│  0  0  0 0 0 0 0    │  0  0  0 0 0 0 0    │1 1 1 1 1 1 1│
│  0  0  0 0 0 0 0    │  0  0  0 0 0 0 0    │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 1 1 1│
│ _3 _2 _1 0 1 2 3    │ _3 _2 _1 0 1 2 3    │1 1 1 1 1 1 1│
│  6  3  1 0 0 1 3    │  6  3  1 0 0 1 3    │1 1 1 1 1 1 1│
│_10 _4 _1 0 0 0 1    │_10 _4 _1 0 0 0 1    │1 1 1 1 1 1 1│
└─────────────────────┴─────────────────────┴─────────────┘
```

GKP5.9 shows a more interesting discrepancy that is somewhat exacerbated by the use of the GKP definition:

```
   L5_9=:+/@([ (]!+) Ei@])
   R5_9=:] ! >:@+
   (L5_9 T i);(R5_9 T i);(L5_9 T i)=(R5_9 T i)
┌─────────────────────┬────────────────────┬─────────────┐
│_1 1 0 1 _1  0  0    │ 0  0 0 1 _1  0  0  │0 0 1 1 1 1 1│
│ 0 1 0 1  0  0  0    │ 0  0 0 1  0  0  0  │1 0 1 1 1 1 1│
│ 1 1 0 1  1  1  1    │ 1  1 1 1  1  1  1  │1 1 0 1 1 1 1│
│ 2 1 0 1  2  3  4    │_2 _1 0 1  2  3  4  │0 0 1 1 1 1 1│
│ 3 1 0 1  3  6 10    │ 1  0 0 1  3  6 10  │0 0 1 1 1 1 1│
│ 4 1 0 1  4 10 20    │ 0  0 0 1  4 10 20  │0 0 1 1 1 1 1│
│ 5 1 0 1  5 15 35    │ 0  0 0 1  5 15 35  │0 0 1 1 1 1 1│
└─────────────────────┴────────────────────┴─────────────┘
   L5_9a=:+/@([ (] GKP5_1 +) Ei@])
   R5_9a=:] GKP5_1 >:@+
   (L5_9a T i);(R5_9a T i);(L5_9a T i)=(R5_9a T i)
┌─────────────────────┬──────────────────┬─────────────┐
│_1 1 0 1 _1  0  0    │0 0 0 1 _1  0  0  │0 0 1 1 1 1 1│
│ 0 1 0 1  0  0  0    │0 0 0 1  0  0  0  │1 0 1 1 1 1 1│
│ 1 1 0 1  1  1  1    │0 0 0 1  1  1  1  │0 0 1 1 1 1 1│
│ 2 1 0 1  2  3  4    │0 0 0 1  2  3  4  │0 0 1 1 1 1 1│
│ 3 1 0 1  3  6 10    │0 0 0 1  3  6 10  │0 0 1 1 1 1 1│
│ 4 1 0 1  4 10 20    │0 0 0 1  4 10 20  │0 0 1 1 1 1 1│
│ 5 1 0 1  5 15 35    │0 0 0 1  5 15 35  │0 0 1 1 1 1 1│
└─────────────────────┴──────────────────┴─────────────┘
```

```
   L5_10=:(+/@([ ! Ei@]))"0
   R5_10=: !&>:
   (L5_10 T i);(R5_10 T i); (L5_10 T i)=(R5_10 T i)
┌─────────────┬────────────────┬─────────────┐
│0 0 0 0 0 0 0│ 1 _1 0 0 0 0 0 │0 0 1 1 1 1 1│
│0 0 0 0 0 0 0│ 0  1 0 0 0 0 0 │1 0 1 1 1 1 1│
│0 0 0 0 0 0 0│ 1  1 1 1 1 1 1 │0 0 0 0 0 0 0│
│2 1 0 1 2 3 4│_2 _1 0 1 2 3 4 │0 0 1 1 1 1 1│
│1 0 0 0 1 3 6│ 3  1 0 0 1 3 6 │0 0 1 1 1 1 1│
│0 0 0 0 0 1 4│_4 _1 0 0 0 1 4 │0 0 1 1 1 1 1│
│0 0 0 0 0 0 1│ 5  1 0 0 0 0 1 │0 0 1 1 1 1 1│
└─────────────┴────────────────┴─────────────┘
   L5_10a=: +/@([ GKP5_1 Ei@])
   R5_10a=: GKP5_1&>:
   (L5_10a T i);(R5_10a T i); (L5_10a T i)=(R5_10a T i)
┌─────────────┬────────────────┬─────────────┐
│0 0 0 0 0 0 0│ 0  0 0 0 0 0 0 │1 1 1 1 1 1 1│
│0 0 0 0 0 0 0│ 0  0 0 0 0 0 0 │1 1 1 1 1 1 1│
│0 0 0 0 0 0 0│ 1  1 1 1 1 1 1 │0 0 0 0 0 0 0│
│2 1 0 1 2 3 4│_2 _1 0 1 2 3 4 │0 0 1 1 1 1 1│
│1 0 0 0 1 3 6│ 3  1 0 0 1 3 6 │0 0 1 1 1 1 1│
│0 0 0 0 0 1 4│_4 _1 0 0 0 1 4 │0 0 1 1 1 1 1│
│0 0 0 0 0 0 1│ 5  1 0 0 0 0 1 │0 0 1 1 1 1 1│
```

We will use GKP5.22 to illustrate the treatment of relations among the binomial coefficients. Using the forms **(m,n) L (r,s)** and **(m,n) R (r,s)** for the left and right limbs, we will first define functions that permit the use of notation similar to that of GKP:

```
   m=: {.@[
   n=: {:@[
   r=: {.@]
   s=: {:@]
```

GKP uses m+k and n-k to suggest pairs of values that sum to m+n. For this we will define and use the following functions:

```
   L1=: Ei@(m+n)
   L2=: |.@L1
   a=: 2 3
   b=: 4 5
   a (m ; n ; r ; s ; L1 ; L2) b
+-+-+-+-+-----------+-----------+
|2|3|4|5|0 1 2 3 4 5|5 4 3 2 1 0|
+-+-+-+-+-----------+-----------+
```

The left and right limbs of GKP5.22 may now be defined as follows:

```
   L=: (L1 ! r) X (L2 ! s)
     X=: +/ . *
   R=: (m+n) ! (r+s)
   a(R ; L ; (L1 ! r) ; L2 ! s) b
+---+---+-----------+-------------+
|126|126|1 4 6 4 1 0|1 5 10 10 5 1|
+---+---+-----------+-------------+

   <"2 (R ; L ; [ ; ])"1/~ i. 3 2
+---------------+-------------------+---------------+
|+-+-+---+---+  |+---+---+---+---+  |+-+-+---+---+  |
||1|1|0 1|0 1|  ||0  |0  |2 3|0 1|  ||0|0|4 5|0 1|  |
|+-+-+---+---+  |+---+---+---+---+  |+-+-+---+---+  |
||5|5|0 1|2 3|  ||1  |1  |2 3|2 3|  ||0|0|4 5|2 3|  |
|+-+-+---+---+  |+---+---+---+---+  |+-+-+---+---+  |
||9|9|0 1|4 5|  ||126|126|2 3|4 5|  ||1|1|4 5|4 5|  |
|+-+-+---+---+  |+---+---+---+---+  |+-+-+---+---+  |
+---------------+-------------------+---------------+
```

The entire theorem can be defined more concisely as follows:

```
   T5_22=: +/@(* |.)/@(] !~/ Ei@(+/)@[) -: !&(+/)
   T5_22"1/~i.3 2
1 1 1
1 1 1
1 1 1
```

B. POWER SERIES

Polynomials. The binomials are called *coefficients* because they often serve as the coefficients `c` in the expression `+/c*x^i.#c` that defines a polynomial:

```
   poly=: +/@([ * ] ^ i.@#@[) " 1 0
   c=: 1 3 3 1 [ d=: 1 2 1
   x=: 0 1 2 3 4
   (c poly x);((x+1)^3);(d poly x);((x+1)^2)
```

```
┌──────────────┬──────────────┬────────────┬────────────┐
│1 8 27 64 125│1 8 27 64 125│1 4 9 16 25│1 4 9 16 25│
└──────────────┴──────────────┴────────────┴────────────┘
   c p. x
1 8 27 64 125
```

The last expression above shows the use of the primitive polynomial function `p.`.

The functions `pp` and `ps` are called the *polynomial product* and *polynomial sum*, whose uses are illustrated as follows:

```
   pp=: +//.@(*/)
   ps=: +/@,:
   (c pp d);(c ps d)
┌────────────────┬─────────┐
│1 5 10 10 5 1│2 5 4 1│
└────────────────┴─────────┘
   ((c pp d)&p.;(c&p. * d&p.);(c ps d)&p.;(c&p. + d&p.)) x
┌──────────────────┬──────────────────┬──────────────┬──────────────┐
│1 32 243 1024 3125│1 32 243 1024 3125│2 12 36 80 150│2 12 36 80 150│
└──────────────────┴──────────────────┴──────────────┴──────────────┘
```

The binomial coefficients can be produced by applying the product function `pp` to the argument `1 1` (the coefficients of a polynomial equivalent to the function `1:+]`).

```
   1 1 pp 1 1
1 2 1
   pp/ 5#,:1 1
1 5 10 10 5 1
   (pp/\ 5#,:1 1) ; (1 1&pp^:(i.6) 1)
┌─────────────┬─────────────┐
│1 1  0  0 0 0│1 0  0  0 0 0│
│1 2  1  0 0 0│1 1  0  0 0 0│
│1 3  3  1 0 0│1 2  1  0 0 0│
│1 4  6  4 1 0│1 3  3  1 0 0│
│1 5 10 10 5 1│1 4  6  4 1 0│
│             │1 5 10 10 5 1│
└─────────────┴─────────────┘
```

The product `*/x-r` is equivalent to a polynomial in the argument `x`, and the function `pir` is called a polynomial in terms of roots. Thus:

```
   pir=: */@:-~"1 0
   r=: 3 1 5 2
   (r pir x);(r&pir x);(r&pir r)
┌───────────┬───────────┬───────┐
│30 0 0 0 _6│30 0 0 0 _6│0 0 0 0│
└───────────┴───────────┴───────┘
```

The last result illustrates that the elements of `r` are the *zeros* or *roots* of `r&pir`.

The coefficients of a polynomial equivalent to `r&pir` are obtained as a product (`pp`) of the factors `(-r),.1`:

```
   cfr=: pp/@(-,.1:)
   c=: cfr r
   c ; (c p. x) ; (r pir x)
┌───────────────┬───────────┬───────────┐
│30 _61 41 _11 1│30 0 0 0 _6│30 0 0 0 _6│
└───────────────┴───────────┴───────────┘
   cfr\ -1 1 1 1 1
1 1  0  0 0 0
1 2  1  0 0 0
1 3  3  1 0 0
1 4  6  4 1 0
1 5 10 10 5 1
```

Power series. The expression `(g i.n)&p.` defines an `n`-term polynomial with coefficients generated by the function `g`; it is called a *power series*. For example:

```
   g=: %@!
   g i. n=: 5
1 1 0.5 0.1666667 0.04166667
   (g i. n)&p.
1 1 0.5 0.1666666666666666 0.04166666666666666&p.
   ((g i. n)&p. ,: ^) x=: (i. 5) % 2
1 1.64844 2.70833 4.39844       7
1 1.64872 2.71828 4.48169 7.38906
```

We define a power series conjunction as follows:

```
   psc=: ].@(i.@([."_)) p. ]
   5 psc g x
1 1.64844 2.70833 4.39844 7
```

Taylor's Theorem (cited on page 163 of GKP) states that coefficient `k` of a power series approximation to a function `f` is the `k`th derivative of `f`, evaluated at `0` and divided by factorial `k`.

The phrase `f t. k` uses the adverb `t.` to produce the `k`th Taylor coefficient of the function `f`. For example:

```
   ^ t. 2
0.5
   ]ce=: ^ t. i. 7
1 1 0.5 0.1666667 0.04166667 0.008333333 0.001388889
   %ce
1 1 2 6 24 120 720
   ce p. x=: 0.2*i.6
1 1.2214 1.49182 1.82211 2.22549 2.71806
   ^x
1 1.2214 1.49182 1.82212 2.22554 2.71828
   sin=: 1&o. [. cos=: 2&o.
   csin=: sin t. i.7
   ccos=: cos t. i.7
   csin,:ccos
0 1    0 _0.1666667          0 0.008333333            0
1 0 _0.5         0 0.04166667           0 _0.001388889
```

A wide range of functions (including the circular and hyperbolic functions, exponential, logarithm, and all polynomials defined by a finite list of coefficients) are representable as Taylor's series in the sense that for any specified argument `x` in the relevant domain and specified tolerance `tol`, a value of `k` can be chosen such that the difference `((f t. i.k)&p.á|@- f) x` does not exceed `tol`. Taylor coefficients can be used to explore the polynomial equivalents of identities such as sin squared plus cos squared equals one. Thus:

```
   pp=: +//.@(*/)
   0.3 ": csin pp csin
0.000  0.000  1.000  0.000  _0.333  0.000  0.044  0.000  _0.003
     0.000 0.000 0.000 0.000
   (csin pp csin) p. x
0 0.0394695 0.1516469 0.3188274 0.5146589 0.7084028
   *:@sin x
0 0.0394695 0.1516466 0.3188211 0.5145998 0.7080734
   0.3 ": (pp~csin)+(pp~ccos)
1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
     0.000 0.000 0.000 0.000
```

```
   1 2 1&p. t. i.6
1 2 1 0 0 0
   pp~ 1 2 1&p. t. i.6
1 4 6 4 1 0 0 0 0 0 0
```

The coefficients corresponding to one polynomial atop another can be obtained as follows:

```
   atop=: [ X unit , pp/\@table
     table=: >@(<:@#@[ $ <@])
     pp=: +//.@(*/)
     unit=: #@[ {. 1:
     X=: +/ . *
   ]c=: a atop b [ a=: 1 2 1 [ b=: 1 3 3 1
4 12 21 22 15 6 1
   (a&p.)@(b&p.) t. i. 7
4 12 21 22 15 6 1
   c p. x=: i.5
4 81 784 4225 15876
   (a&p.)@(b&p.) x
4 81 784 4225 15876
   atop f.
[ +/ .* (#@[ {. 1:) , (+//.@(*/)/\@(>@(<:@#@[ $ <@])))
```

Linear functions. `f` is said to be a linear vector function if `f(c+d)` equals `(f c)+(f d)` for any vectors `c` and `d`. For example:

```
   l1=: +/@(] * *:@i.@#)"1
   l2=: +/\@(] * *:@i.@#)"1
   c=: 3 1 4 2 [ d=: 1 3 3 1
   (l1 c);(l1 d);((l1 c)+(l1 d));(l1 c+d)
┌──┬──┬──┬──┐
│35│24│59│59│
└──┴──┴──┴──┘
   ,.&.>(l2 c);(l2 d);((l2 c)+(l2 d));(l2 c+d)
┌──┬──┬──┬──┐
│ 0│ 0│ 0│ 0│
│ 1│ 3│ 4│ 4│
│17│15│32│32│
│35│24│59│59│
└──┴──┴──┴──┘
```

Linearity can be expressed more succinctly by the tautology:

```
   taut=: l1@:+ -: +&l1
   (c l1@:+ d);(c +&l1 d);(c taut d)
┌──┬──┬─┐
│59│59│1│
└──┴──┴─┘
```

Any linear function can be expressed as a matrix product in the form `X & rm`, where `rm` is a suitably chosen matrix, or, equivalently, in the form `lm & X`, where `lm` is the transpose of `rm`. The matrix `rm` may be obtained by simply applying the linear function to the rows of an identity matrix. Thus:

```
   Id=: i. =/ i.
   lm=: |: rm=: l2"1 Id # c
   h=: lm & X
   s=: X & rm
   ,.&.> (Id#c);rm;lm;(lm X c);(h c);(c X rm);(s c);(l2 c)
┌───────┬───────┬──────────┬───┬──┬──┬──┬──┐
│1 0 0 0│0 0 0 0│1 7 49 343│892│ 0│ 0│ 0│ 0│
│0 1 0 0│0 1 1 1│1 5 25 125│358│ 1│ 1│ 1│ 1│
│0 0 1 0│0 0 4 4│1 3  9  27│ 96│17│17│17│17│
```

```
|0 0 0 1|0 0 0 9|1 2  4   8| 37|35|35|35|35|
```

Since the sum of polynomials `c&p.` and `d&p.` can also be expressed as the polynomial `(c+d)&p.`, it appears that polynomials are somehow linear in their coefficients. For example:

```
   x=: 7 5 3 2
   ,.&.>((c p.x);(d p.x);((c p.x)+(d p.x));((c+d)p.x))
+---+---+----+----+
|892|512|1404|1404|
|358|216| 574| 574|
| 96| 64| 160| 160|
| 37| 27|  64|  64|
+---+---+----+----+
```

This may be shown more clearly as follows:

```
   g=: p.&x                    A linear function

   ,.&.>((g c);(g d);((g c)+(g d));(g c+d);(lm=:|:g"1 Id#c))
+---+---+----+----+----------+
|892|512|1404|1404|1 7 49 343|
|358|216| 574| 574|1 5 25 125|
| 96| 64| 160| 160|1 3  9  27|
| 37| 27|  64|  64|1 2  4   8|
+---+---+----+----+----------+
```

The matrix `lm` may be recognized as the Vandermonde matrix `x ^/ i. #x`.

C. RATIONAL FUNCTIONS

If the functions `f` and `g` in the quotient `q=: f%g` are polynomials, then `q` is called a *rational* function. The conjunction `over` defined below produces a rational function in terms of its coefficients. Thus:

```
   over=: ([.&p.) % (].&p.)
   c=: 1 4 6 4 1
   d=: 1 2 1
   c over d
1 4 6 4 1&p. % 1 2 1&p.
   c over d z=: i. 6
1 4 9 16 25 36
   d over c z
1 0.25 0.1111111 0.0625 0.04 0.02777778
   ]ser=: c over d t. i. 10
1 2 1 0 0 0 0 0 0 0
   d pp ser
1 4 6 4 1 0 0 0 0 0 0 0
```

The result `ser` contains the first ten elements of the power series that represents the rational `c over d`.

We will now define an adverb `REC` such that `c REC k` gives the first `k` coefficients of the reciprocal polynomial `1 over c`:

```
   1 over 1 1 t. i. 10
1 _1 1 _1 1 _1 1 _1 1 _1
   REC=: 1 : '1 over x. t. @i.'
   1 1 REC 6
1 _1 1 _1 1 _1
   1 _1 _1 REC 10
0 1 1 2 3 5 8 13 21 34                    Fibonacci numbers
```

The basis for this rather surprising result of Fibonacci numbers (previously defined recursively) is examined in Chapter 7 of GKP.

Partial fraction expansion. A rational function `f%g` can be expressed as `f * %@g`. If the polynomial `g` is expressed as a product of its roots `r`, then the reciprocal `%*/r` can also be expressed as a weighted sum of the reciprocals of the individual roots, using a process called *partial fraction expansion*.

For example, if `r=: a,b,c`, then the reciprocal `t=: 1 % */a,b,c` may be expressed as the sum `s=: (x%a)+(y%b)+(z%c)` for suitable values of `x`, `y`, and `z`. Multiplying both `t` and `s` by `a*b*c` and equating the results shows that the sum `(x*b*c)+(y*a*c)+(z*a*b)` must equal `1`. The values of `x`, `y`, and `z` can therefore be obtained as a solution of this linear equation. Thus:

```
   'abc'=: 2 3 4
   (a,b,c);(%*/a,b,c);(d=: (b*c),(a*c),(a*b))
```

2 3 4	0.04166667	12 8 6

```
   ]'xyz'=: %. d                             See §J
0.04918033 0.03278689 0.02459016
   (x,y,z);(+/d*x,y,z)                       Shows that d is a solution
```

0.04918033 0.03278689 0.02459016	1

```
   (x%a)+(y%b)+(z%c)                         Sum equals reciprocal product
0.04166667
```

The rational fraction expansion can be re-expressed more neatly and more generally in terms of the vector `r`. In particular, the multipliers of `x`, `y`, and `z` may each be seen to be the products over all *except* the corresponding element of `r`, a result that is given by the expression `1 */\. r`. Thus:

```
   PFE=: %.@(1: */\. ])                      Partial fraction expansion
   ]q=: PFE r=: 2 3 4 5
0.008987418 0.005991612 0.004493709 0.003594967
   (q%r);(+/q%r);(%*/r)
```

0.004493709 0.001997204 0.001123427 0.0007189934	0.008333333	0.008333333

D. MULTINOMIALS

Just as `(x+y)^t` can be expressed as a weighted sum of products of powers of `x` and `y`, so can the power `(+/v)^t` be expressed as a weighted sum of products of powers of the elements of the list `v`:

```
   sqsum=:(+/v)^2 [ v=: 2 3 5 [ c=: 2 2 2 1 1 1
   t=: 0 1 1,1 0 1,1 1 0,0 0 2,0 2 0,:2 0 0
   t;(v^"1 t);(,.p);(,.c);(+/c*p=:*/"1 v^"1 t);sqsum
```

0 1 1 1 0 1 1 1 0 0 0 2 0 2 0 2 0 0	1 3 5 2 1 5 2 3 1 1 1 25 1 9 1 4 1 1	15 10 6 25 9 4	2 2 2 1 1 1	100	100

The multinomial (MN) exponents of the elements of `v` must sum to `t`, and the table of exponents and list of coefficients may be obtained as follows (See page 168 ff. of GKP):

```
mc=: !@] % */"1@!@me                    MN coefficients
me=: ](]#~ [ = +/"1@]) all          MN exponents
  all=: (] #: i.@(*/))@base              All in base
    base=: [ # ] + 1: [. m=: 3 [. t=: 2
 (m base t);(m all t);(m me t);(,. m mc t)
```

3 3 3	0 0 0 0 0 1 0 0 2 0 1 0 0 1 1 0 1 2 0 2 0 0 2 1 0 2 2 1 0 0 1 0 1 1 0 2 1 1 0 1 1 1 1 1 2 1 2 0 1 2 1 1 2 2 2 0 0 2 0 1 2 0 2 2 1 0 2 1 1 2 1 2 2 2 0 2 2 1 2 2 2	0 0 2 0 1 1 0 2 0 1 0 1 1 1 0 2 0 0	1 2 1 2 2 1

Binomials may be treated as special cases of the multinomial:

```
be=: 2&me [. bc=: 2&mc
 (be 4); (,. bc 4)
```

0 4 1 3 2 2 3 1 4 0	1 4 6 4 1

Autonomous definitions can be obtained by using the fix adverb `f.`. Thus:

```
   me f.
] (] #~ [ = +/"1@]) (] #: i.@(*/))@([ # ] + 1:)
```

E. SUBFACTORIALS AND STATIONARY POINTS

On page 194 GKP introduces the notion of a subfactorial function:

> A permutation is called a derangement if it moves every item, and the number of derangements of n objects is sometimes denoted by the symbol ‘n¡’, read as “n subfactorial.”

The following page of GKP presents several definitions, three of which we will paraphrase as follows, using the alternating sum (`-/`) to obviate the powers of negative one:

```
SF1=: |@(-/)@(!@Ei * Ei ! ])
SF2=: ! * -/@%@!@Ei
SF3=: [: <.0&= + %@2: + ! % ^@1:
```

```
   (SF1,SF2,SF3)"0 i.9
    1     1     1
    0     0     0
    1     1     1
    2     2     2
    9     9     9
   44    44    44
  265   265   265
 1854  1854  1854
14833 14833 14833
```

The elements of a permutation that do not move appear as single cycles in its cycle representation. For example:

```
   p=: 6 3 2 1 4 0 5
   C. p
┌─┬───┬─┬─────┐
│2│3 1│4│6 5 0│
└─┴───┴─┴─────┘
```

and `p` is not a derangement because elements `2` and `4` do not move. We will use the cycle representation to determine the number of stationary points in a permutation as follows:

```
   NSP=: +/@(1: = #@>@C.)"1
   D=: 0: = NSP
   (NSP , D) p
2 0
   allp=: (i.@! A. i.)"0              All permutations of order of argument
   ([ ; C. ; ,.@NSP ; ,.@D) allp 3
┌─────┬──────────────┬─┬─┐
│0 1 2│┌─────┬───┬─┐ │3│0│
│0 2 1││0    │1  │2│ │ │0│
│1 0 2│├─────┼───┼─┤ │1│0│
│1 2 0││0    │2 1│ │ │ │1│
│2 0 1│├─────┼───┼─┤ │1│1│
│2 1 0││1 0  │2  │ │ │ │0│
│     │├─────┼───┼─┤ │0│ │
│     ││2 0 1│   │ │ │ │ │
│     │├─────┼───┼─┤ │0│ │
│     ││2 1 0│   │ │ │ │ │
│     │├─────┼───┼─┤ │1│ │
│     ││1    │2 0│ │ │ │ │
│     │└─────┴───┴─┘ │ │ │
└─────┴──────────────┴─┴─┘
```

We will use `D` to define a direct count of derangements as follows:

```
   (SF4=: +/@D@allp) i. 8
1 0 1 2 9 44 265 1854
```

The distribution of the number of stationary points in a set of permutations is given by:

```
   DNSP=: +/@(NSP =/ Ei@{:@$)
   DNSP@allp i. 7
  1   0   0  0  0 0 0   This is the table of the function h that follows GKP5.59
  0   1   0  0  0 0 0
  1   0   1  0  0 0 0
  2   3   0  1  0 0 0
  9   8   6  0  1 0 0
 44  45  20 10  0 1 0
265 264 135 40 15 0 1
```

F. FALLING AND RISING FACTORIALS (STOPES)

We will now re-examine the matter of falling factorials, introduced in §2E (finite calculus). The expression x(x-1)(x-2) ... (x-m+1) is used in GKP2.43 to define a falling factorial of order m. The adverb `g` defined below provides a set of functions `s g` such that `x s g t` yields a `t`-element grid beginning with `x` and changing in steps of size `s`. The product over such a grid is called a factorial function in GKP. Thus:

```
   g=: 1 : '[ + x."_ * i.@]'
   x=: 6
   t=: 4
```

```
   (*/y);(y=: x _1 g t)                 Falling factorial
┌───┬───────┐
│360│6 5 4 3│
└───┴───────┘
   (*/y);(y=: x 1 g t)                  Rising factorial
┌────┬───────┐
│3024│6 7 8 9│
└────┴───────┘
   (*/y);(y=: x 0 g t)                  The power function
┌────┬───────┐
│1296│6 6 6 6│
└────┴───────┘
   (*/y);(y=: x 1r2 g t)                Fractional step
┌──────┬───────────┐
│2047.5│6 6.5 7 7.5│
└──────┴───────────┘
   (*/y);(y=: x 0j1 g t)                Imaginary step
┌─────────┬─────────────┐
│900j1260 │6 6j1 6j2 6j3│
└─────────┴─────────────┘
```

These products are "variants" of the power function `^`, and in **J** are provided by the fit conjunction `!.`, using `^!.s`, or the equivalent adverb `stope=: ^!.`. Thus:

```
   stope=: ^!.
   (x ^!._1 t);(x _1 stope t);(x 0j1 stope t)
┌───┬───┬────────┐
│360│360│900j1260│
└───┴───┴────────┘
```

Because the factors descend or ascend in steps like a stope in a mine, we call `^!.s` a *stope* function, and extend the notion of a polynomial as a weighted sum of powers to *stope polynomials* that are weighted sums of stope functions. Such a polynomial is provided by the variant `p.!.s`, where `s` specifies the step size. We therefore define an adverb whose (left) argument specifies the step size in the stope function used. Thus:

```
   sp=: 1 : 'p.!.x.'
   _1 sp
p.!._1
    c=: 1 4 6 4 1 [ d=: 1 15 25 10 1
   x=: 0 1 2 3
   (c 0 sp x);(c 1 sp x);(c _1 0 1 sp x);(d _1 sp x)
┌───────────┬────────────┬──────────┬───────────┐
│1 16 81 256│1 65 261 685│ 1   1   1│1 16 81 256│
│           │            │ 5  16  65│           │
│           │            │21  81 261│           │
│           │            │73 256 685│           │
└───────────┴────────────┴──────────┴───────────┘
```

As illustrated by the agreement of the first and last panels above, two different stope polynomials can be made to agree by a suitable choice of coefficients. The coefficients `d` are obtained as a linear function of `c`, using the transformation `t` developed in the discussion of the finite calculus in §2E. This may be re-expressed in terms of the conjunction `!.` as follows:

```
   s=: (i. ^/ i.) %. (i. ^!._1/ i.)
   t=: s@# X ]
   d=: t c
   ,.&.>((s 4);(%.s 4);c;d;(c p. x);(d p.!._1 x))
┌───────┬──────────┬─┬──┬───┬───┐
│1 0 0 0│1 0  0  0│1│ 1│  1│  1│
│0 1 1 1│0 1 _1  2│4│15│ 16│ 16│
│0 0 1 3│0 0  1 _3│6│25│ 81│ 81│
│0 0 0 1│0 0  0  1│4│10│256│256│
│       │          │1│ 1│   │   │
└───────┴──────────┴─┴──┴───┴───┘
```

We will now extend Vandermonde matrices to stopes in a manner analogous to the extension of polynomials, and use them to define a conjunction `from` for the coefficients of the linear transformation illustrated above:

```
   from=: 2 : '^!.y./~@i. %. ^!.x./~@i.'
   _1 from 0
^!.0/~@i. %. ^!._1/~@i.
```

Falling factorial coefficients from power coefficients

```
   m=: _1 from 0 # c
   d=: m +/ . * c
   ,.&.>(x ; c ; (c p. x) ; d ; (d p.!._1 x) ; m)
┌─┬─┬───┬──┬───┬─────────┐
│0│1│  1│ 1│  1│1 0 0 0 0│
│1│4│ 16│15│ 16│0 1 1 1 1│
│2│6│ 81│25│ 81│0 0 1 3 7│
│3│4│256│10│256│0 0 0 1 6│
│ │1│   │ 1│   │0 0 0 0 1│
└─┴─┴───┴──┴───┴─────────┘
```

As remarked in §2E, the matrices for the transformations `_1 from 0` and `0 from _1` are simply related to Stirling numbers of the first and second kind. Stirling numbers are extensively discussed in Chapter 6 of GKP, and references are made to related transformations such as `_1 from 1` (falling factorial from rising factorial):

```
   _1 from 1 (7)
1 0 0 0  0   0    0
0 1 2 6 24 120  720
0 0 1 6 36 240 1800
0 0 0 1 12 120 1200
0 0 0 0  1  20  300
0 0 0 0  0   1   30
0 0 0 0  0   0    1
```

G. HYPERGEOMETRIC FUNCTIONS

GKP5.76 defines the hypergeometric as a sum over the products of two lists for all `k` not less than zero. The first list is the quotient of products over the rising factorials of two list parameters denoted by `a` and `b`, and the second is the powers of the argument `z` divided by corresponding factorials.

To confine the matter to finite lists, we will introduce a further argument `t`, and treat only the values of `k` specified by `i. t`. The definitions will be used in the form `t a hy b z`, where `hy` is a conjunction. Thus:

```
   X=: +/ . *
   rf=: 1 : '(,x.)"_ ^!.1/ i.@['       The ,x. treats scalars as lists
   L1=: 2 : 'x.rf %&(*/) y.rf'          A conjunction
   L2=: (i.@[ ^~ ]) % (!@i.@[)
   hy=: L1 X L2                         A conjunction
```

For example:

```
     a=: 2 3 5 [ b=: 6 5
     t=: 4 [ z=: 7
   a L1 b
(2 3 5"_ ^!.1/ i.@[) %&(*/) (6 5"_ ^!.1/ i.@[)
   ]y1=: t a L1 b z
1 1 1.71429 4.28571
   ]y2=: t L2 z
1 7 24.5 57.1667
   y1 X y2
295
   t a hy b z
295
   t a hy b "0 i. 8
1 3.57143 12.1429 31 64.4286 116.714 192.143 295
```

Other cases noted in GKP can be tested as follows:

```
   (10) 1 hy 1 " 0 i. 6
1 2.71828 7.38871 20.0634 54.1541 143.689
   ^ i. 6
1 2.71828 7.38906 20.0855 54.5982 148.413
   (10) '' hy '' " 0 i. 6
1 2.71828 7.38871 20.0634 54.1541 143.689
   ]y=: 2r10 * i. 6
0 0.2 0.4 0.6 0.8 1
   (10) 1 1 hy 1 " 0 y
1 1.25 1.66649 2.48488 4.46313 10
   % 1-y
1 1.25 1.66667 2.5 5 _
   (10) 1r2 1 hy 1 " 0 y
1 1.11803 1.29096 1.57863 2.15355 3.52394
   %(1-y)^1r2
1 1.11803 1.29099 1.58114 2.23607 _
   (10) _1r2 1 hy 1 y
1 0.8944272 0.7745982 0.6325735 0.450624 0.1854706
   %(1+y)^1r2
1 0.9128709 0.8451543 0.7905694 0.745356 0.7071068
   y * (10) 1 1 hy 2 " 0 y
0 0.2231435 0.5108196 0.915551 1.57887 2.92897
   ^. 1+y
0 0.1823216 0.3364722 0.4700036 0.5877867 0.6931472
```

The primitive hypergeometric conjunction `H.` produces a rank-0 dyadic function equivalent to `a hy b`, and a monadic case that is its limit for an unlimited number of terms. For example:

```
   10 '' H. '' i. 6
1 2.71828 7.38871 20.0634 54.1541 143.689
   '' H. '' 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
   (i. 8) '' H. '' 3
0 1 4 8.5 13 16.375 18.4 19.4125
   '' H. '' 3
20.0855
   ((^.@-. % -) -: 1 1 H. 2) x=:j./0.01*>:?2 40$50
1
```

Abramowitz and Stegun [7] provide an extensive collection of hypergeometric identities, using the *Pochhammer* notation $(a)_n$ [referred to on page 48 of GKP] and the notation F(a, b; c; z) for `(a,b) H. c z`. For example:

```
   L15_1_7=: 1r2 1r2 H. 3r2 @-@*:                    Page 556 of A&S
   C15_1_7=: (%:@>:@*: * 1 1 H. 3r2)@-@*:
   R15_1_7=: % * ^.@(] + %:@>:@*:)
   ]z=: 1r10 * 1 + i. 3 3
0.1 0.2 0.3
0.4 0.5 0.6
0.7 0.8 0.9
(L15_1_7 , C15_1_7 ,: R15_1_7) z
0.9983408 0.9934506 0.9855768
0.9750883 0.9624237 0.9480415
0.9323808        _.        _.

0.9934359 0.9749374 0.9478269
 0.916862 0.8873108 0.8640129
0.8506062        _.        _.

0.9983408 0.9934506 0.9855768
0.9750883 0.9624237 0.9480415
0.9323808 0.9158353  0.898741
```

As shown in the definition of the conjunction `hy`, noun arguments to the equivalent primitive conjunction `H.` serve as arguments to the rising factorial function; *verb* arguments to `H.` act directly upon the integers `i. n`. For example:

```
   k=: i. 5
   ]z=: k % 10
0 0.1 0.2 0.3 0.4
   (1: H. 1: z) ,: ^ z                               Exponential
1 1.10517 1.2214 1.34986 1.49182
1 1.10517 1.2214 1.34986 1.49182
   sinh=: 5&o.                                       Hyperbolic sine
   2&| k                                             Alternate 0 and 1
0 1 0 1 0
   (2&| H. 1: z) ,: sinh z
0 0.1001668 0.201336 0.3045203 0.4107523
0 0.1001668 0.201336 0.3045203 0.4107523
   sin=: 1&o.                                        Sine

   (2&| H. 1: &. j. z) ,: sin z                    Under mult by 0j1
0 0.09983342 0.1986693 0.2955202 0.3894183
0 0.09983342 0.1986693 0.2955202 0.3894183
```

H. AGGREGATES AND DIFFERENCES

Applied to a list, the functions `a` and `d` defined below provide aggregates and differences, respectively, and are partial inverses of one another. For example:

```
   a=: +/\
   d=: }. - }:
   x=: ^&2 >: i. 6
   ,.&.> q=:x;(a x);(d x);(d a x);(a d x);(d d x);(d^:3 x)
┌──┬──┬──┬──┬──┬─┬─┐
│ 1│ 1│ 3│ 4│ 3│2│0│
│ 4│ 5│ 5│ 9│ 8│2│0│
│ 9│14│ 7│16│15│2│0│
│16│30│ 9│25│24│2│ │
│25│55│11│36│35│ │ │
```

36	91					

Since the result of `d x` has fewer elements than `x`, it cannot (as illustrated above) possess a strict inverse. We redefine `d` as the inverse of `a` as follows:

```
d=: a ^: _1
r=:x;(a x);(d x);(d a x);(a d x);(d d x);(d^:3 x)
,.&.> r
```

1 4 9 16 25 36	1 5 14 30 55 91	1 3 5 7 9 11	1 4 9 16 25 36	1 4 9 16 25 36	1 2 2 2 2 2	1 1 0 0 0 0

The effect is to retain the leading element of `x` as the leading element of `d x`. Since both `a` and `d` are linear functions, the (mutually inverse) matrices that represent them are:

```
(a"1 ; d"1) I=: =/~ i. 6
```

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	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 0 0 0 0 0 1

The difference operator of GKP2.42 can be represented by the following adverb:

```
q=: 1 : '-/@x.@(1 0"_ +/ ])'
x=: 1 + i. 6
,.&.>((^&2);(^&2 q);(^&3);(^&3 q);(^&3 q q)) x
```

1 4 9 16 25 36	3 5 7 9 11 13	1 8 27 64 125 216	7 19 37 61 91 127	12 18 24 30 36 42

However, we will instead define a conjunction `DD` such that `f DD s` gives the divided differences with spacing `s`; in particular, `DD 1` will be equivalent to the adverb `q` above:

```
DD=: 2 : '-/@x.@((y.,0)"_ +/ ]) % y."_'
sq=: ^&2
u=:((sq DD 1);(sq DD 1r2);(sq DD _1);(sq DD 1e_8)) x
,.&.> u                          An approximation to the derivative
```

3 5 7 9 11 13	2.5 4.5 6.5 8.5 10.5 12.5	1 3 5 7 9 11	2 4 6 8 10 12

I. GENERATING FUNCTIONS

A dyadic function `d` is said to be an *approximating function* for a function `f` on a given domain if for any specified positive tolerance `t` it is possible to find an integer `n` such that the difference `(n&d |@- f) x` is less than `t` for any argument `x` in the domain. For example, `%@!k` is the `k`th Taylor coefficient for the exponential, and the polynomial `%@!@[ p. ]`is an approximating function for it. For example:

```
   d=: (%@!@i.@[ p. ])"0
   (7 d x) ,: ^ x
1 2.71806 7.35556 19.4125 48.5556
1 2.71828 7.38906 20.0855 54.5982
   (7 8 9 d/ x) , ^ x
1 2.71806 7.35556 19.4125 48.5556
1 2.71825 7.38095 19.8464 51.8063
1 2.71828  7.3873 20.0092 53.4317
1 2.71828 7.38906 20.0855 54.5982
```

If the polynomial `h@i.@[ p. ]` is an approximating function for `f`, then `h` is said to be a *generating function* for `f`.

The coefficients `e` for the product function `f*g` can be equated to the polynomial product `+//. c */ d` of the coefficients `c` and `d` for `f` and `g`. Thus:

```
   f=: (c=: 1 2 1)&p.
   g=: (d=: 1 3 3 1)&p.
   x=: 0 1 2 3 4
   ,.&.> (f;g;f*g) x
 1   1    1
 4   8   32
 9  27  243
16  64 1024
25 125 3125

   ]e=: c pp d
1 5 10 10 5 1
   e&p. x
1 32 243 1024 3125
```

If the coefficients for the functions `f` and `g` are themselves expressible as functions of their indices, then equating the coefficients for `f*g` with the polynomial product of the coefficients for `f` and `g` can establish interesting relations. For example:

```
   t=: 7
   c=: %@! i. t                 Coefficients for exponential ^
   d=: c* _1^2|i.t              Alternating coeffs for decaying exponential ^@-
   x=: 0.1*0 1 2 3 4
   ,.&.>((c&p.);(d&p.);((c&p.)*(d&p.))) x
      1         1 1
1.10517 0.9048374 1
 1.2214 0.8187308 1
1.34986 0.7408183 1
1.49182 0.6703204 1
   <7.3 ": c */ d               Print 3 decimal places and box
   1.000 _1.000  0.500 _0.167  0.042 _0.008  0.001
   1.000 ¯1.000  0.500 ¯0.167  0.042 ¯0.008  0.001
   0.500 ¯0.500  0.250 ¯0.083  0.021 ¯0.004  0.001
   0.167 ¯0.167  0.083 ¯0.028  0.007 ¯0.001  0.000
   0.042 ¯0.042  0.021 ¯0.007  0.002 ¯0.000  0.000
   0.008 ¯0.008  0.004 ¯0.001  0.000  0.000  0.000
   0.001 ¯0.001  0.001 ¯0.000  0.000  0.000  0.000
   7.3 ": +//. c */ d
  1.000  0.000  0.000  0.000  0.000  0.000  0.000
```

```
      0.000  0.000  0.000  0.000  0.000  0.000
   7.3 ": c pp d
  1.000  0.000  0.000  0.000  0.000  0.000  0.000
      0.000  0.000  0.000  0.000  0.000  0.000
```

The case of Vandermonde's convolution (GKP5.27 and page 198) may be treated thus:

```
   bc=: !&                              Adverb for binomial coefficients
   (t=: 7) bc i. 10
1 7 21 35 35 21 7 1 0 0
   ]c=: t bc Ei t
1 7 21 35 35 21 7 1
   ]x=: 0 1 2 3 4 - 3
_3 _2 _1 0 1
   (c&p. x);((x+1)^t);((c&p. * c&p.) x)
┌────────────────┬────────────────┬──────────────────┐
│_128 _1 0 1 128│_128 _1 0 1 128│16384 1 0 1 16384│
└────────────────┴────────────────┴──────────────────┘
   (x+1)^(t+t)
16384 1 0 1 16384
   c */ c
 1   7  21   35   35  21   7  1
 7  49 147  245  245 147  49  7
21 147 441  735  735 441 147 21
35 245 735 1225 1225 735 245 35
35 245 735 1225 1225 735 245 35
21 147 441  735  735 441 147 21
 7  49 147  245  245 147  49  7
 1   7  21   35   35  21   7  1
   ]e=: +//. c */ c
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
   c pp c                                    Left side of GKP5.27
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
   e&p. x
16384 1 0 1 16384
   (t+t) bc i.t+t+1                          Right side of GKP5.27
1 14 91 364 1001 2002 3003 3432 3003 2002 1001
    364 91 14 1
```

To aid in visualizing the behaviour of the diagonal summation performed by `+//.` on the product table `c */ d`, we will use the diagonal adverb `/.` with boxed literal arguments, and a "catenation under box" function instead of summation and product:

```
   c=: ;:'a0 a1 a2 a3' [ d=: ;:'b0 b1 b2 b3'
   cub=: ,&.>
   c;<c cub d
┌─────────────┬─────────────────────┐
│┌──┬──┬──┬──┐│┌────┬────┬────┬────┐│
││a0│a1│a2│a3│││a0b0│a1b1│a2b2│a3b3││
│└──┴──┴──┴──┘│└────┴────┴────┴────┘│
└─────────────┴─────────────────────┘
   (c cub/ d) ; <(cub/. c cub/ d)
┌─────────────────────┬─────────────────────┐
│┌────┬────┬────┬────┐│┌────┬────┬────┬────┐│
││a0b0│a0b1│a0b2│a0b3│││a0b0│    │    │    ││
│├────┼────┼────┼────┤│├────┼────┼────┼────┤│
││a1b0│a1b1│a1b2│a1b3│││a0b1│a1b0│    │    ││
│├────┼────┼────┼────┤│├────┼────┼────┼────┤│
││a2b0│a2b1│a2b2│a2b3│││a0b2│a1b1│a2b0│    ││
│├────┼────┼────┼────┤│├────┼────┼────┼────┤│
││a3b0│a3b1│a3b2│a3b3│││a0b3│a1b2│a2b1│a3b0││
│└────┴────┴────┴────┘│├────┼────┼────┼────┤│
```

a1b3	a2b2	a3b1	
a2b3	a3b2		
a3b3			

J. NOTATION

Polynomial in terms of roots and multiplier. The function `pir` defined in §B can be replaced by the primitive `p.`. The phrase `(<r) p. x` evaluates the polynomial with roots `r`; that is, it is equivalent to `*/x-r`. Moreover, the phrase `(m;r) p. x` is equivalent to `m * (<r) p. x`.

Taylor coefficients. The adverb `t.` produces a function for Taylor coefficients. For example:

```
   ^ t. i.5
1 1 0.5 0.1666667 0.04166667

   3 1 4&p. t. i.5
3 1 4 0 0
```

Format. The phrase `0.7 ": n` produces the literal list that represents `n` to `7` decimal digits. The phrase `12.7 ": m` allots a width of `12` to each column of a matrix `m`. For example:

```
   0.7 ": 1p1
3.1415927

   12.7 ": m=: o. i. 2 3
   0.0000000   3.1415927   6.2831853
   9.4247780  12.5663706  15.7079633

   $ 12.7 ": m=: o. i. 2 3
2 36
```

Reflection in unit circle or sphere. The degenerate case of matrix inverse for a list `a` (treated as a one-column matrix) is the solution of a linear equation with coefficients `a` and result `1`. Geometrically, the result is the reflection of the point with coordinates `a` in a unit circle, sphere, or hyper-sphere. Thus:

```
   ]r=: %. a=: 1 2 3
0.07142857 0.1428571 0.2142857
   +/r*a
1
```

Cycles. `C. p` yields the boxed cycles of a permutation `p`. The power of a single cycle is the number of its elements, and the power of a permutation is the LCM of the powers of the included cycles. For example:

```
   p=: 11 16 0 10 13 1 12 18 19 6 7 9 14 3 15 2 17 5 4 8
   ] cy=: C. p
```

15 2 0 11 9 6 12 14	17 5 1 16	18 4 13 3 10 7	19 8

```
   ]n=: #&>cy
8 4 6 2

   ]pow=: *./n
24

   ]i=: i. # p
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

   p&{ ^:pow i
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Infix and Outfix. The phrase `n f\ x` applies `f` to each infix of `x` of length `n`, and `n f\. x` applies it to outfixes. For example:

```
   2 <\ x=: 2 3 4 5 6
┌───┬───┬───┬───┐
│2 3│3 4│4 5│5 6│
└───┴───┴───┴───┘
   2 <\. x=: 2 3 4 5 6
┌─────┬─────┬─────┬─────┐
│4 5 6│2 5 6│2 3 6│2 3 4│
└─────┴─────┴─────┴─────┘

   (2 */\ x) ; (2 */\. x)
┌──────────┬────────────┐
│6 12 20 30│120 60 36 24│
└──────────┴────────────┘
```

Chapter 6
SPECIAL NUMBERS

A. STIRLING NUMBERS

Stirling numbers of the first kind concern the number of ways to array `n` objects in `k` cycles; those of the second kind concern the number of ways to partition a set of `n` things into `k` non-empty subsets.

By making and examining tables of the first few values, GKP arrive at the following recursive definitions (GKP6.8, 3):

```
   S1=:(k=0:)`((n $:&<: k) + (n-1:)* (n-1:)$:k) @. (n>0:) " 0
   S2=:(k=0:)`((n $:&<: k) +  k * (n-1:)$:k) @. (n>0:) " 0
   n=: [ [. k=: ]
```

Tables 245 and 244 of GKP are therefore produced as follows:

```
   i=: i. 10
   i (S1/;S2/) i
┌──────────────────────────────────────────────────┬──────────────────────────────────────┐
│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 1   0    0    0    0    0   0  0 0│
│0     1      1      0     0     0    0   0  0 0│0 1   1    0    0    0    0   0  0 0│
│0     2      3      1     0     0    0   0  0 0│0 1   3    1    0    0    0   0  0 0│
│0     6     11      6     1     0    0   0  0 0│0 1   7    6    1    0    0   0  0 0│
│0    24     50     35    10     1    0   0  0 0│0 1  15   25   10    1    0   0  0 0│
│0   120    274    225    85    15    1   0  0 0│0 1  31   90   65   15    1   0  0 0│
│0   720   1764   1624   735   175   21   1  0 0│0 1  63  301  350  140   21   1  0 0│
│0  5040  13068  13132  6769  1960  322  28  1 0│0 1 127  966 1701 1050  266  28  1 0│
│0 40320 109584 118124 67284 22449 4536 546 36 1│0 1 255 3025 7770 6951 2646 462 36 1│
└──────────────────────────────────────────────────┴──────────────────────────────────────┘
```

As remarked in §2F, the matrix of Stirling numbers of the second kind (and its inverse) provide the transformations of coefficients for the falling factorial polynomial `p.!._1`. As remarked in GKP, the removal of the factor `k` from the recursion defining `S2` yields a recursion for the binomial coefficients. Thus:

```
   bc=:(k=0:)`((n $:&<: k) + (n-1:)$:k) @. (n>0:) " 0
   bc/~ i.5
1 0 0 0 0
1 1 0 0 0
1 2 1 0 0
1 3 3 1 0
1 4 6 4 1
```

The foregoing recursions are all *double* in the sense that recursion is applied to each of the two arguments. In *Representations of Recursion* [6] it is shown that equivalent single recursions on vectors may be defined as follows:

```
   S1v=: 1:`([ S1r $:@<:) @. * " 0
     S1r=: (0:,]) + <:@[ * ],0: " 0
   S2v=: 1:`(Ei S2r $:@<:) @. * " 0
     S2r=: (0:,]) + [ * ],0: " 0
   bcv=: 1:`(bcr@$:@<:) @. * " 0
     bcr=: (0:,]) +(],0:)
   (S1v;S2v;bcv) i. 5
┌──────────┬─────────┬─────────┐
│1 0  0 0 0│1 0 0 0 0│1 0 0 0 0│
│0 1  0 0 0│0 1 0 0 0│1 1 0 0 0│
│0 1  1 0 0│0 1 1 0 0│1 2 1 0 0│
│0 2  3 1 0│0 1 3 1 0│1 3 3 1 0│
│0 6 11 6 1│0 1 7 6 1│1 4 6 4 1│
└──────────┴─────────┴─────────┘
```

As remarked on page 248 of GKP, “... Stirling subset numbers are the coefficients of factorial powers that yield ordinary powers”. Consequently, the conjunction:

```
f=: from=: 2 : '^!.y./~@i. %. ^!.x./~@i.'
```

from §5F may be used to produce many of the transformations discussed in GKP6.11-33. For example:

```
f=: from=: 2 : '^!.y./~@i. %. ^!.x./~@i.'
Sn10=: _1 f 0 [. S0n1=: 0 f _1 [. S01=: 0 f 1 [. S1n1=: 1 f _1
  Thr=: ] * 0.1&^@[ <: |@]            Threshold for rounding to zero
1&Thr@|:&.> (Sn10 ; S0n1 ; S01 ; S1n1) 6
```

```
┌──────────────┬─────────────────────┬─────────────────┬──────────────────────────┐
│1 0  0  0  0 0│1  0   0  0   0 0│1  0  0  0  0 0│1   0    0   0   0 0│
│0 1  0  0  0 0│0  1   0  0   0 0│0  1  0  0  0 0│0   1    0   0   0 0│
│0 1  1  0  0 0│0 _1   1  0   0 0│0  1  1  0  0 0│0  _2    1   0   0 0│
│0 1  3  1  0 0│0  2  _3  1   0 0│0  2  3  1  0 0│0   6   _6   1   0 0│
│0 1  7  6  1 0│0 _6  11 _6   1 0│0  6 11  6  1 0│0 _24   36 _12   1 0│
│0 1 15 25 10 1│0 24 _50 35 _10 1│0 24 50 35 10 1│0 120 _240 120 _20 1│
└──────────────┴─────────────────────┴─────────────────┴──────────────────────────┘
```

The successive panels show the matrix of Stirling numbers of the second kind, its inverse (whose magnitudes equals those of the first kind), numbers of the first kind, and the transformation to rising factorials from falling factorials (GKP6.14).

Just as GKP determined recursive definitions for the Stirling numbers by exploring linear relations among entries in a non-recursively defined table, so we might explore linear relations between a non-recursively defined table such as `|: Sn10` and some suitably shifted variant of it. For example:

```
sh=: }:"1@}:@Ad
  Ad=: 1:(<0 0)} 0&,@(0&,.)
a=: |: Sn10 7
1&Thr&.> (] ; sh ; ] %. sh) a
```

```
┌─────────────────┬─────────────────┬─────────────┐
│1 0  0  0  0  0 0│1 0 0  0  0  0 0│1 0 0 0 0 0 0│
│0 1  0  0  0  0 0│0 1 0  0  0  0 0│0 1 0 0 0 0 0│
│0 1  1  0  0  0 0│0 0 1  0  0  0 0│0 1 1 0 0 0 0│
│0 1  3  1  0  0 0│0 0 1  1  0  0 0│0 0 2 1 0 0 0│
│0 1  7  6  1  0 0│0 0 1  3  1  0 0│0 0 0 3 1 0 0│
│0 1 15 25 10  1 0│0 0 1  7  6  1 0│0 0 0 0 4 1 0│
│0 1 31 90 65 15 1│0 0 1 15 25 10 1│0 0 0 0 0 5 1│
└─────────────────┴─────────────────┴─────────────┘
```

The results of §5F provide the basis for the following non-recursive functions for the Stirling numbers:

```
s1nr=: |@|:@(^!._1/~ %. ^/~)@i."0
s2nr=:|:@(^/~ %. ^!._1/~)@i."0
```

The identity expressed by GKP6.15 uses the binomial coefficients, and the two limbs may be expressed as follows:

```
Bc=: i. !/ i.
X=: +/ . *
L6_15=: s2nr@>:
R6_15=: Ad@(|:@Bc X s2nr)
```

Thus:

```
(s2nr; Bc ; L6_15 ; R6_15) 5
```

```
┌─────────┬─────────┬─────────────┬─────────────
│1 0 0 0 0│1 1 1 1 1│1 0 0 0 0 0│1 0 0 0 0 0
│0 1 0 0 0│0 1 2 3 4│0 1 0 0 0 0│0 1 0 0 0 0
│0 1 1 0 0│0 0 1 3 6│0 1 1 0 0 0│0 1 1 0 0 0
│0 1 3 1 0│0 0 0 1 4│0 1 3 1 0 0│0 1 3 1 0 0
│0 1 7 6 1│0 0 0 0 1│0 1 7 6 1 0│0 1 7 6 1 0
```

```
│          │          │0 1 15 25 10 1│0 1 15 25 10 1│
└──────────┴──────────┴──────────────┴──────────────┘
```

We will now use these results to define a number of tautologies given in GKP. Note that the falling factorial and the rising factorial (denoted in GKP by $x^{\underline{n}}$ and a similar overbar) are here denoted by `^!._1` and `^!.1`:

```
   ff=: ^!._1 [. rf=: ^!.1
   XA=: -/ . *
   GKP6_14=: ff -: _1&^@] * -@[ rf ]
   GKP6_15=: (s2nr@>: -: Ad@(|:@Bc X s2nr))"0
   GKP6_16=: (s1nr@>: -: Ad@(|:@Bc X~ s1nr))"0
   GKP6_17=: (s2nr -: |@(|:@Bc XA 1 1&}.@s2nr@>:))"0
   GKP6_31=: ((i. =/ i.) -: |@(s2nr XA s1nr))"0
```

For example:

```
   x=: 2 3 5
   (x GKP6_14 x),(GKP6_15,GKP6_16,GKP6_17,GKP6_31) 5
1 1 1 1 1
```

It may be noted that all of the summations indicated in GKP6.15 and 6.16 are, in `GKP6_15` and `GKP6_16`, performed by the matrix product `X=: +/ . *`. Moreover, each use of `Ad` to bring the right limb up to the shape of the left could, if we ignore the matter of the agreement of the trivial leading row and leading column, be replaced by dropping the leading row and column from the left limb. For example:

```
   GKP6_15a=: (1 1&}.@s2nr@>: -: |:@Bc X s2nr)"0
```

Expressions such as the `(-1)`$^{t-s}$ that occurs in GKP6.17 and 6.31 are avoided by using the "alternating" matrix product `XA=: -/ . *` instead of `X`, as illustrated in Theorems `GKP6_17` and `GKP6_31`.

B. EULERIAN NUMBERS

The recursion for Eulerian numbers defined by GKP6.35-36 can be expressed using the top and bottom arguments within angles as left and right arguments, respectively. Thus:

```
E=: (((k + 1:)*((n-1:)E k))+((n-k)*((n-1:)E(k-1:))))`(k= 0:)@.c"0
  c=: (n <: 0:) +. (k < 0:)
    n=: [
    k=: ]
   (i. E/ i.) 5
1  0  0 0 0
1  0  0 0 0
1  1  0 0 0
1  4  1 0 0
1 11 11 1 0
```

If each occurrence of `E` in the definition is replaced by `$:` (self-reference), the resulting definition can be assigned to any name. Moreover, many of the forks may be replaced by more compact expressions; for example, `(k + 1:)` can be replaced by `>:@k`. Thus:

```
   F=: ((>:@k*<:@n $: k)+-*$:&<:)`(k = 0:)@.c"0
   (i. F/ i.) 5
1  0  0 0 0
1  0  0 0 0
1  1  0 0 0
1  4  1 0 0
1 11 11 1 0
```

Such a double recursion can consume enormous amounts of time and space, and we define an equivalent single recursion that produces an entire vector result as follows:

```
   EV=: 1:`(Ei r $:@<:)@.*"0
```

```
     r=: (|.@[ * 0:,]) + (>:@[ * ],0:)

   EV i. 12
     0      0       0       0        0       0       0      0    0 0 0
1    0      0       0       0        0       0       0      0    0 0 0
1    1      0       0       0        0       0       0      0    0 0 0
1    4      1       0       0        0       0       0      0    0 0 0
1   11     11       1       0        0       0       0      0    0 0 0
1   26     66      26       1        0       0       0      0    0 0 0
1   57    302     302      57        1       0       0      0    0 0 0
1  120   1191    2416    1191      120       1       0      0    0 0 0
1  247   4293   15619   15619     4293     247       1      0    0 0 0
1  502  14608   88234  156190    88234   14608     502      1    0 0 0
1 1013  47840  455192 1310354  1310354  455192   47840   1013    1 0 0
1 2036 152637 2203488 9738114 15724248 9738114 2203488 152637 2036 1 0
```

Worpitzky's identity (GKP6.37) involves the binomial coefficients which we saw in the preceding chapter were produced by the function `!~`. We will test the identity after first fixing a definition of the function `F` so as to have no dependence on the definitions of `n` and `k` as left and right arguments:

```
   EU=: F f.
     n=: {.@]
     x=: [
     k=: {:@]
   WORP=: (n EU/ k) */ (x+/k) !~/ n
    a=: 3 4 5
    b=: 0 1 2 3 4 ,: 0 1 2 3 4
    q=: a WORP b
    $q
5 5 3 5 5
   j=: 1 3;2;0 4
   r=: j |: q
```

Run together axes 1 and 3 as well as 0 and 4

```
   $r
5 3 5
   +/r
1 3  9  27  81
1 4 16  64 256
1 5 25 125 625
   a (x ^/ n) b
1 3  9  27  81
1 4 16  64 256
1 5 25 125 625
```

The second order Eulerian functions may be defined similarly, using GKP6.41:

```
    n=:[ [. k=: ]
   SOE=: ((>:@k*<:@n$:k)+(+:@[->:@k)*$:&<:)`(k=0:)@.c"0
   0 1 2 3 SOE/ 0 1 2 3
1 0 0 0
1 0 0 0
1 2 0 0
1 8 6 0
```

Representations of Recursion [6] gives the following single recursion for second-order Eulerian numbers:

```
   fv =: 1: ` (>:@Ei fr $:@<:) @. *
     fr =: ((-~ +:@#) * 0:,]) + ([ * ],0:)
   fv"0 i.12
1   0    0     0     0     0     0     0     0     0     0 0
1   0    0     0     0     0     0     0     0     0     0 0
1   2    0     0     0     0     0     0     0     0     0 0
```

```
1    8      6         0         0         0         0         0         0         0         0 0
1   22     58        24         0         0         0         0         0         0         0 0
1   52    328       444       120         0         0         0         0         0         0 0
1  114   1452      4400      3708       720         0         0         0         0         0 0
1  240   5610     32120     58140     33984      5040         0         0         0         0 0
1  494  19950    195800    644020    785304    341136     40320         0         0         0 0
1 1004  67260  1.0625e6 5.7655e6 1.24401e7 1.10263e7 3.73392e6    362880         0         0 0
1 2026 218848 5.32616e6 4.4765e7 1.55357e8 2.38905e8 1.62187e8  4.4339e7  3.6288e6         0 0
1 4072 695038 2.52439e7 3.1437e8 1.64838e9  4.0027e9 4.64216e9 2.50748e9 5.68356e8 3.99168e7 0
```

C. HARMONIC NUMBERS

The harmonic numbers are defined as sums of the reciprocals of integers beginning at `1`:

```
   H=: +/@%@Ai " 0 [. Ai=: >:@i.
   H 3
1.83333
   H i.8
0 1 1.5 1.83333 2.08333 2.28333 2.45 2.59286
   0 1 3r2 11r6 25r12 137r60 49r20 363r140          Ratios shown in GKP
0 1 1.5 1.83333 2.08333 2.28333 2.45 2.59286
```

We will first define functions to obtain the continued fraction representation of a ratio such as `137r60`, and from it the pair `137 60` that represents it. Finally, we will define a function to add rationals represented as two-element vectors.

The expression `3+%7+%15+%1` yields the value of the *continued fraction* represented by the vector `3 7 15 1`. Thus:

```
   3+%7+%15+%1
3.14159
   Cf=: (+%)/
   Cf c=: 3 7 15 1
3.14159
```

Successive prefixes of a vector provide improving approximations to the complete continued fraction, successive pairs bounding the complete result below and above :

```
   Cf\ c
3 3.14286 3.14151 3.14159
```

Any rational number can be represented exactly by a continued fraction, and any non-negative number can be approximated by one. The continued fraction representation may be obtained by repeated use of the following step:

```
   step=: }: , (<. , 1: % ] - <.)@{:        Replace last element by the
   ]a=: 34%13                                integer part, followed by the
2.61538                                      reciprocal of the fractional part
   step a
2 1.625
   step step a
2 1 1.6
   step^:0 1 2 3 4 5 6 a
2.61538      0   0       0   0 0           0
      2 1.625   0       0   0 0           0
      2     1 1.6       0   0 0           0
      2     1   1 1.66667   0 0           0
      2     1   1       1 1.5 0           0
      2     1   1       1   1 2           0
      2     1   1       1   1 2 _5.6295e14              Large reciprocal
                                                                       of
approximate
   b=: step^:5 a                                         zero signals end
   b ; (Cf b) ; a                                        of useful process
┌───────────┬───────┬───────┐
│2 1 1 1 1 2│2.61538│2.61538│
└───────────┴───────┴───────┘
```

The function `step` can be incorporated in a recursion with an agenda that tests for a large reciprocal fractional part, and also (to make it usable for arguments such as `pi=: o. 1` which would otherwise never terminate) for an upper limit on the number of elements in the result. Thus:

```
   Cfex=: }:`($:@step)@.((16"_ > #) *. 1e8&>@|@{:)
   Cfex a
2 1 1 1 1 2
   Cfex pi=: o. 1
3 7 15 1 292 1 1 1 2 1 3 1 14 3 3
   (] ; step ; step@step ; Cfex ; Cf@Cfex) a=: 137 % 60
┌───────┬─────────┬───────────┬───────────┬───────┐
│2.28333│2 3.52941│2 3 1.88889│2 3 1 1 7 1│2.28333│
└───────┴─────────┴───────────┴───────────┴───────┘
```

The rational number that represents a fraction is produced by the following recursively-defined function:

```
   Ratio=: Rat@(;&1 0)@Cfex " 0
   Rat=: >@{:`($:@rstep) @. (0:<#@>@{.)
   rstep=: }:@F ; (L@L + L@F * F@L) , (F@L)
     F=: >@{.
     L=: >@{:
   (rstep@(;&1 0) ;Ratio ; %/@Ratio ; Cf) 2 1 5
┌─────────┬────┬───────┬───────┐
│┌───┬───┐│17 6│2.83333│2.83333│
││2 1│5 1││    │       │       │
│└───┴───┘│    │       │       │
└─────────┴────┴───────┴───────┘
   Ratio 137r60
137 60
```

We will now define a function (non-recursively) to add rationals represented as two-element vectors:

```
   plus=: red@(num,den)
     red=: ] % +./            Divide by GCD to reduce to lowest terms
     num=: [ +/@:* |.@]       Sum of products of numerators with denominators
     den=: *&{:               Product of denominators
   2 4 plus 7 8
11 8
   (i. 5 2) ; plus/\i. 5 2
┌───┬────────┐
│0 1│   0   1│
│2 3│   2   3│
│4 5│  22  15│
│6 7│ 244 105│
│8 9│1012 315│
└───┴────────┘

   rec=: 1: ,. ]              Catenate leading 1 to represent reciprocals
   Ai=: >:@i.                 Augmented indices
   tab=: plus/\@rec@Ai        Table of rational sums
   (,.@Ai ; rec@Ai ; tab ; ,.@(%/"1@tab) ; ,.@(+/@%\@Ai)) 12
┌──┬────┬───────────┬───────┬───────┐
│ 1│1  1│     1    1│      1│      1│
│ 2│1  2│     3    2│    1.5│    1.5│
│ 3│1  3│    11    6│1.83333│1.83333│
│ 4│1  4│    25   12│2.08333│2.08333│
```

```
│ 5│1  5│  137    60│2.28333│2.28333│
│ 6│1  6│   49    20│   2.45│   2.45│
│ 7│1  7│  363   140│2.59286│2.59286│
│ 8│1  8│  761   280│2.71786│2.71786│
│ 9│1  9│ 7129  2520│2.82897│2.82897│
│10│1 10│ 7381  2520│2.92897│2.92897│
│11│1 11│83711 27720│3.01988│3.01988│
│12│1 12│86021 27720│3.10321│3.10321│
└──┴────┴───────────┴───────┴───────┘
```

The fact that the harmonic series diverges is established by summing over successive groups of `2^i.k` elements. Such grouping can be controlled by a boolean vector as illustrated below:

```
   u=: 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0
   u < ;. 1 Ai 15                    Boxing of groups
┌─┬───┬───────┬─────────────────────┐
│1│2 3│4 5 6 7│8 9 10 11 12 13 14 15│
└─┴───┴───────┴─────────────────────┘
   ]q=: u +/@% ;. 1 Ai 15            Sum over reciprocal of groups
1 0.8333333 0.7595238 0.7253719
   (+/q) , H 15                      Total of all groups equals the harmonic
3.31823 3.31823
```

The boolean vector for such a cut can be generated as illustrated below:

```
   (cutpoints=: ;@(<@}:@#:"0@(2&^^:2@i.))) 5
1 1 0 1 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
```

D. HARMONIC SUMMATION

A sum of the harmonic numbers may be expressed as a simple product and difference. This is done in GKP6.67, whose left and right limbs may be expressed as follows:

```
   L=: +/@:H@i.
   R=: (] * H) - ]
   L"0 i.7
0 0 1 2.5 4.33333 6.41667 8.7
   R"0 i.7
0 0 1 2.5 4.33333 6.41667 8.7
```

The right limb may be defined more neatly by `R=: ]*H-1:`. Weighted sums (such as sums of products with the binomial coefficients) can be expressed as a matrix product in the manner used in the tautologies at the end of §A.

E. BERNOULLI NUMBERS

A sum of powers of the form `+/@: ((i.@n) ^ m)` can also be expressed as a polynomial in `n`, and the coefficients are called *Bernoulli Numbers*. We will approach this matter by using the adverb `CPA` (used similarly in §2E) to determine the coefficients. We will further use the function `Ratio` of §C to determine these coefficients as rational numbers, in an effort to identify patterns in their formation.

We begin by defining a function `S` such that `m S n` produces a table of such sums. Thus:

```
   S=: +/@: ((i.@n) ^ m)"0
   m=: [ [. n=: ]
   (i.8) S/ (i.8)
0 1 2  3  4   5   6    7
0 0 1  3  6  10  15   21
0 0 1  5 14  30  55   91
0 0 1  9 36 100 225  441
0 0 1 17 98 354 979 2275
```

```
0 0 1  33  276  1300  4425  12201
0 0 1  65  794  4890 20515  67171
0 0 1 129 2316 18700 96825 376761
```

The adverb `CPA` is used in the expression `f CPA n` to give the coefficients of the best fit polynomial of order `n` to the function `f`. Thus:

```
   CPA=: (@Ei) %. ^/~@Ei [. Ei=: i.@>:
   7&S i. 8
0 0 1 129 2316 18700 96825 376761
   ]c=: 7&S CPA 8
4.61121e_6 0.008617654 0.06273281 0.01870953 _0.3002615
    0.002196546 0.5830169 _0.499976 0.1249993
   0 ": c p. i. 8
0 0 1 129 2316 18700 96825 376761
```

The ratios that represent a fraction are given by the following suite of functions:

```
   Ratio=: Rat@(;&1 0)@Cfex " 0
     Rat=: >@{:`($:@rstep) @. (0:<#@>@{.)
       rstep=: }:@F ; (L@L + L@F * F@L) , (F@L)
         F=: >@{.
         L=: >@{:
     Cfex= .}:`($:@step)@.(((10^8)&>@|@{:)*.(16"_ > #))
       step=: }: , (<. , 1: % ] - <.)@{:
```

Applying `Ratio` to the coefficients representing a single row of the sums produced by the function `S` we have the ratios `r5` as follows:

```
   c5=: 5&S CPA 6
   ]r5=: |: Ratio c5
0 0 _1 0  5 _1 1
1 1 12 1 12  2 6
   (%/d5) p. i. 6
0 _1.38778e_17 1 33 276 1300
```

GKP provides a table of such ratios that begins as shown below, and recommends examining them to seek a pattern:

```
   R=: |:@:Ratio
   C=: CPA
   r0=: R 0&S C 6 [ r1=: R 1&S C 6 [ r2=: R 2&S C 6
   r3=: R 3&S C 6 [ r4=: R 4&S C 6 [ r5=: R 5&S C 6
   ,. r0;r1;r2;r3;r4;r5
+-------------------+
|0  1 0 0  0 0 0    |
|1  1 1 1  1 1 1    |
+-------------------+
|0 _1 1 0  0 0 0    |
|1  2 2 1  1 1 1    |
+-------------------+
|0 1 _1 1  0 0 0    |
|1 6  2 3  1 1 1    |
+-------------------+
|0 0 1 _1  1 0 0    |
|1 1 4  2  4 1 1    |
+-------------------+
|0 _1 0 1 _1 1 0    |
|1 30 1 3  2 5 1    |
+-------------------+
|0 0 _1 0  5 _1 1   |
|1 1 12 1 12  2 6   |
+-------------------+
```

F. FIBONACCI NUMBERS

The recursion for generating Fibonacci numbers given by GKP6.102 may be expressed as follows:

```
   F=: 0&~:`(+/@:F@(<:@<: , <:))@.(1&<)"0

   F i. 15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

We will illustrate the treatment of the theorems following GKP6.102 with the case of GKP6.108, first defining a dyadic function `G` such that `(n G k)=(F n+k)`:

```
   G=: F@+

   L6_108=: G
   R6_108=: (F@] * 1&G@[) + (_1&G@] * F@[)

   <"2 (L6_108 , R6_108)"0/~i.6
```

0 0	1 1	1 1	2 2	3 3	5 5
1 1	1 1	2 2	3 3	5 5	8 8
1 1	2 2	3 3	5 5	8 8	13 13
2 2	3 3	5 5	8 8	13 13	21 21
3 3	5 5	8 8	13 13	21 21	34 34
5 5	8 8	13 13	21 21	34 34	55 55

The expression `_1&G@]` used in the right limb may invoke `F` with a negative argument, and the identity as stated holds only because `F` produces `1` for negative arguments.

G. CONTINUANTS

The continued fraction, commonly written in the form of GKP6.126, can also be written in the form `(+%)/` used in §C for rational approximations to numbers such as π. Thus:

```
   3+%7+%15+%1
3.14159

   cf=: (+%)/

   cf c=: 3 7 15 1
3.14159

   cf\ c
3 3.14286 3.14151 3.14159
```

The *continuant polynomial* may be defined directly from GKP6.127 as follows:

```
   K=: */`((K@}: * {:) + (K@}:@}:))@.(# > 1:)

   (K 6#1) , (F 6+1)
13 13
```

Illustration of GKP6.128

The products over all possible selections of elements from a list can be expressed as illustrated below:

```
   b=: #: i. 2 ^ # x=: 2 3 4

   b ; (x ^"1 b) ; (,. */"1 x^"1 b)
```

0 0 0	1 1 1	1
0 0 1	1 1 4	4
0 1 0	1 3 1	3
0 1 1	1 3 4	12
1 0 0	2 1 1	2
1 0 1	2 1 4	8
1 1 0	2 3 1	6
1 1 1	2 3 4	24

Those rows of the complete boolean array `b` that correspond to “striking out adjacent pairs” are those in which each unbroken string of zeros has an even number of elements. These can be identified by the selection function `sel`, defined and illustrated below:

```
   sel=: (0:=+./@(2: | (# ;._1 @(1&,))))"1
   b=: #: i. 2 ^ # x=: 2 3 4 5

   |: 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
   sel b
1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1
   prd=: */"1 pow=: x ^"1 q=: (sel b) # b
   q ; pow ; (,. prd) ; (+/prd) ; (K x)
┌───────┬───────┬───┬───┬───┐
│0 0 0 0│1 1 1 1│  1│157│157│
│0 0 1 1│1 1 4 5│ 20│   │   │
│1 0 0 1│2 1 1 5│ 10│   │   │
│1 1 0 0│2 3 1 1│  6│   │   │
│1 1 1 1│2 3 4 5│120│   │   │
└───────┴───────┴───┴───┴───┘
```

The equality of the last two results illustrates GKP6.129

The treatment of further theorems will be illustrated by GKP6.133. For the right limb we will define a dyadic function `R` whose left argument will specify the *partition point* denoted by m in GKP:

```
   R=: (K@{. * K@}.) + K@(<:@[ {. ]) * K@(>:@[ }. ])

   2 R y=: 2 3 4 5 6 7           Right limb with partition point at 2
6961
   K y                           Left limb
6961
   1 2 3 4 5 R"0 1 y
6961 6961 6961 6961 6961
   0 6 R"0 1 y
28066 7933
```

These last results (that correspond to degenerate partitions with no elements in one of the partitions) are anomalous, and merit further examination by the reader.

The relation between `K` and continued fractions expressed in GKP6.135 may be illustrated as follows:

```
   (K ; K@}. ; (K % K@}.) ; (+%)/) y
┌────┬────┬───────┬───────┐
│6961│3015│2.30879│2.30879│
└────┴────┴───────┴───────┘
```

H. NOTATION

Fix. The adverb `f.` *fixes* the definition to which it is applied, recursively substituting its definition for each component verb used in the definition.

Chapter 7
GENERATING FUNCTIONS

A. DOMINO THEORY AND CHANGE

The idea of a generating function (GKP §5.4) as a polynomial of variable degree whose coefficients are generated by a function was introduced in §5I, and illustrated by Taylor series.

In this chapter GKP uses the problem of tiling a rectangle with 1-by-2 and 2-by-one rectangles (called horizontal and vertical dominoes) to introduce a generalization of the idea.

We will use the following boxed arrays to provide graphic displays of tilings analogous to those shown in GKP:

```
   N=: <i.1 0 [ H=: 1 2$<'-' [ V=: 2 1$<'|'
```

```
   ]A=: V,.H,H              A vertical tile followed by a horizontal atop a horizontal
┌─┬─┬─┐
│|│-│-│
├─┼─┼─┤
│|│-│-│
└─┴─┴─┘
```

```
   ]B=: N ,. V ,. H , H     Prefaced by the null denoted by | in GKP
┌┬─┬─┬─┐
││|│-│-│
├┼─┼─┼─┤
││|│-│-│
└┴─┴─┴─┘
```

GKP denotes a power (or repetition) of a pattern using an exponent as follows:

```
   ┌─┬─┐ 2
   │-│-│
   └─┴─┘
```

and uses it to denote either horizontal or vertical replication. We define an analogous set of functions as follows:

```
   plh=: ,.                 Plus (catenation) horizontally
   plv=: ,                  Plus (catenation) vertically
   poh=: plh/@(] # ,:@[)    Power (replication) horizontally
   pov=: plv/@(] # ,:@[)    Power (replication) vertically
   tih=: poh~               Times (replication) horizontally
   tiv=: pov~               Times (replication) vertically
   V plh V
┌─┬─┐
│|│|│
├─┼─┤
│|│|│
└─┴─┘
   ]block=: V plh (H plv H)
┌─┬─┬─┐
│|│-│-│
├─┼─┼─┤
│|│-│-│
└─┴─┴─┘
   (block poh 2) ; (block pov 2) ;<(2 tiv block)
┌─────────────┬───────┬───────┐
│┌─┬─┬─┬─┬─┬─┐│┌─┬─┬─┐│┌─┬─┬─┐│
││|│-│-│|│-│-│││|│-│-│││|│-│-││
│├─┼─┼─┼─┼─┼─┤│├─┼─┼─┤│├─┼─┼─┤│
││|│-│-│|│-│-│││|│-│-│││|│-│-││
│└─┴─┴─┴─┴─┴─┘│├─┼─┼─┤│├─┼─┼─┤│
│             ││|│-│-│││|│-│-││
│             │├─┼─┼─┤│├─┼─┼─┤│
│             ││|│-│-│││|│-│-││
│             │└─┴─┴─┘│└─┴─┴─┘│
└─────────────┴───────┴───────┘
   3 tih block pov 2
┌─┬─┬─┬─┬─┬─┬─┬─┬─┐
```

|	-	-	|	-	-	|	-	-
|	-	-	|	-	-	|	-	-
|	-	-	|	-	-	|	-	-
|	-	-	|	-	-	|	-	-

The first six terms of the expression for T on page 309 of GKP illustrate the combined use of these facilities:

```
   N plh V plh (V poh 2) plh (H pov 2) plh (V poh 3) plh
       (2 tih V plh H pov 2)
```

|	|	|	|	-	-	|	|	|	|	-	-	|	-	-
|	|	|	|	-	-	|	|	|	|	-	-	|	-	-

GKP suggests that the patterns may be treated as if they are commutative so, for example, the horizontal tiles (represented by the columns `1 2` and `4 5` and `7 8` of `R` as defined below) may be collected at the tail end. Thus:

```
   ]R=: V plh (H plv H) plh V plh (H plv H) plh V plh H plv H
```

|	-	-	|	-	-	|	-	-
|	-	-	|	-	-	|	-	-

```
   0 3 6 1 2 4 5 7 8 {"1 R
```

|	|	|	-	-	-	-	-	-
|	|	|	-	-	-	-	-	-

It would be more perspicuous to group the pairs of columns representing a component pattern so that the permutation would be effected by the index `0 2 4 1 3 5`. This is effected by the following conjunction:

```
   perm=: ;@([.&{)@(].&(<;.1))&.|:
   0 2 4 1 3 5 perm 1 1 0 1 1 0 1 1 0
|:@;@(0 2 4 1 3 5&{@(1 1 0 1 1 0 1 1 0&(<;.1)@|:))
   0 2 4 1 3 5 perm 1 1 0 1 1 0 1 1 0 R
```

|	|	|	-	-	-	-	-	-
|	|	|	-	-	-	-	-	-

Making Change. The number of distinct ways of making change for a given amount in Pennies, in Pennies or Nickels, in Dimes or smaller denominations, and so on through Quarters and Half-dollars is analyzed in GKP to lead to a set of recursively defined functions shown and used below. The term `=&0` that occurs in them accounts for the case of using no coins of a given denomination:

```
   P=: 1"0
   N=: (=&0 + 0:`(P + $:@(-&05))@.(0&<))"0
   D=: (=&0 + 0:`(N + $:@(-&10))@.(0&<))"0
   Q=: (=&0 + 0:`(D + $:@(-&25))@.(0&<))"0
   H=: (=&0 + 0:`(Q + $:@(-&50))@.(0&<))"0
   (>;: '# P PN PND PNDQ PNDQH');(],P,N,D,Q,:H) i. 16
```

#	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
PN	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4
PND	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6
PNDQ	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6
PNDQH	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6

```
   (>;: '# P PN PND PNDQ PNDQH');(],P,N,D,Q,:H) 5*i.11
```

#	0	5	10	15	20	25	30	35	40	45	50
P	1	1	1	1	1	1	1	1	1	1	1
PN	1	2	3	4	5	6	7	8	9	10	11
PND	1	2	4	6	9	12	16	20	25	30	36
PNDQ	1	2	4	6	9	13	18	24	31	39	49
PNDQH	1	2	4	6	9	13	18	24	31	39	50

The last table above can provide some insight. For example, the value `9` in row `PN` of column `40` gives the number of ways for paying forty cents using pennies or nickels. The corresponding value `25` in the next row (using dimes as well) is this value `9` plus the values in each of the columns for the amount decremented by multiples of ten (for dimes), that is, `7` and `5` and `3` and `1`. Similar patterns may be found throughout the table.

B. BASIC MANEUVERS

In GKP7.12 the limit (as `n` approaches infinity) of the polynomial `G=: (g i.n) p. ]` is defined to be the *generating function* for the function `g`. In other words, `g k` is the `k`th element of the Taylor series for `G`.

For example, the functions for the exponential, the sine, and the product of the sine and exponential may be used as generating functions as follows:

```
      gex=: ^ t.
     gsin=: 1&o. t.
   gsinex=: (1&o. * ^) t.
   ]ce=: gex i. 8
1 1 0.5 0.1666667 0.04166667 0.008333333 0.001388889 0.0001984127
   ]cs=: gsin i. 8
0 1 0 _0.1666667 0 0.008333333 0 _0.0001984127
   ]cse=: gsinex i. 8
0 1 1 0.3333333 0 _0.03333333 _0.01111111 _0.001587302
   pp=: +//.@(*/)                 Polynomial product function
   9 {. cs pp ce
0 1 1 0.3333333 0 _0.03333333 _0.01111111 _0.001587302 5.42101e_20
```

In the foregoing, `cs` and `ce` may be recognized as the first few elements of the Taylor series for the exponential and the sine; it should therefore not be surprising that their product under the polynomial product function should agree so well with the coefficients of the product of the sine and exponential. Such manipulations are remarked upon in GKP7.13-7.20.

Polynomial sums, derivatives, and integrals may be treated similarly. Thus:

```
   ps=: +/@,:                     Polynomial sum
   dc=: 1:}.]*i.@#                Polynomial derivative
   ic=: 0:,]%>:@i.@#              Polynomial integral
   sfd=: ("0)(D.1)                Scalar first derivative
   dc cs
1 0 _0.5 0 0.04166667 0 _0.001388889
   (dc cs) p. 0.2*i.6
1 0.9800666 0.921061 0.8253352 0.6967026 0.5402778
   1&o. sfd 0.2*i.6
1 0.9800666 0.921061 0.8253356 0.6967067 0.5403023
```

We will now define an adverb `POW` such that `n f POW x` is the application to `x` of the `n`-term approximation to the power series whose coefficients are generated by the function `f`, and will illustrate its use on the function `f=: !&4` (a function that gives zero for negative arguments):

```
   f=: !&4
   POW=: 1 : 'x.@i.@[ p. ]'
   f POW
f@i.@[ p. ]
   f _2 _1 0 1 2 3 4 5 6
0 0 1 4 6 4 1 0 0
   7 f POW x=: 0 1 2 3 4
1 16 81 256 625
   1 4 6 4 1 p. x
1 16 81 256 625
   (x+1)^4
1 16 81 256 625
   3 f POW x
1 11 33 67 113
   1 4 6 p. x
1 11 33 67 113
```

The function `n&(f POW)` (or, strictly speaking, its limit for large `n`) is the generating function for `f`. We will also define a related adverb that uses a fixed value (`9`) for `n`:

```
   G=: ((@i.) (@9:)) p. ]
   f G x
1 16 81 256 625
```

Table 321 of GKP shows the first few elements of the (coefficients of the) power series for the reciprocals of certain polynomials. The reciprocal is a special case of a rational function, and may be treated in the manner developed in §5C:

```
   over=: ([.&p.) % (].&p.)
   REC=: 1 : '1 over x. t. @i.
```

We will illustrate the matter by the first ten coefficients for a few cases of Table 321:

```
   1 _1 REC 10                            Case:   1/(1-z)
1 1 1 1 1 1 1 1 1 1
   1 1 REC 10                                     1/(1+z)
1 _1 1 _1 1 _1 1 _1 1 _1
   1 0 _1 REC 10                                  1/(1-z²)
1 0 1 0 1 0 1 0 1 0
   1 0 0 0 _1 REC 10                              1/(1-zᵐ) for m=4
1 0 0 0 1 0 0 0 1 0
   ]c=: 1 _1 & pp ^: 2 (1)
1 _2 1
   c REC 10                                       1/(1-z)²
1 2 3 4 5 6 7 8 9 10
   1 _2 REC 10                                    1/(1-2z)
1 2 4 8 16 32 64 128 256 512
   ]c=: 1 _1 & pp ^: (c=: 3) 1
1 _3 3 _1
   c REC 10                                       1/(1-z)ᶜ for c=3
1 3 6 10 15 21 28 36 45 55
   (1, -c=: 3) REC 10                     1/(1-cz)
1 3 9 27 81 243 729 2187 6561 19683
   ]c=: 1 _1 & pp ^: (1 + m=: 3) 1
```

```
1 _4 6 _4 1
```

```
   c REC 10                                  1/(1-z)^(m+1) for m=3
1 4 10 20 35 56 84 120 165 220
```

C. SOLVING RECURRENCES

This section treats a general method for non-recursive definition of a function which is defined by a recursive relationship. The relationship is imposed on a generating function, the resulting equation is solved to express the generating function as a rational function whose expansion as a power series gives its coefficients; that is, the values of the "generated" function. GKP illustrates this method for the Fibonacci numbers, obtaining the reciprocal polynomial with coefficients `1 _1 _1` (a result expressed in GKP as z over 1-z-z^2).

The conjunction `over` and the adverb `REC` of the preceding section may be used to illustrate this:

```
   (0 1 over 1 _1 _1 t. i. 18) ,: (1 _1 _1 REC 18)
0 1 1 2 3 5  8 13 21 34 55  89 144 233 377 610  987 1597
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584
```

If `f` is a function whose result on negative integers is zero, then `f@(-&k)` applied to `i.n` prefaces the results of `f i.n-k` by the results of `f` on the first `k` negative integers. For example:

```
   f=: !&4
   (f ; f@(-&1) ; f@(-&2)) z=: i. 8
```

1 4 6 4 1 0 0 0	0 1 4 6 4 1 0 0	0 0 1 4 6 4 1 0

The fact that `((k#0),c) p. z` is equivalent to `(z^k) * c p. z` implies that (for any function `f` whose result for a negative argument is zero) the generating function satisfies the following relation (which may be executed by first assigning values to the parameters `n`, `k`, and `z`):

```
   n=: 15 [ k=: 5 [ z=: i.8
   (n f@(-&k) POW z) = ((z^k) * (n-k) f POW z)
1 1 1 1 1 1 1 1
```

Ignoring `k` higher order terms, this may be re-expressed as follows:

```
   (n f@(-&k) POW z) = ((z^k) * n f POW z)
```

The following illustrates a general relation between a function and its generating polynomial:

```
   e=: !&3
   pp=: +//.@(*/)
   ps=: +/@,:
   (e pp f) i. 5
1 7 21 35 35 21 7 1 0
   (e ps f) i. 5
2 7 9 5 1
   (e pp f) G z            See §B (a version of POW using a fixed value for n)
1 128 2187 16384 78125 279936 823543 2.09715e6
   (e G z) * (f G z)
1 128 2187 16384 78125 279936 823543 2.09715e6
   (e ps f) G z
2 24 108 320 750 1512 2744 4608
   (e G z) + (f G z)
2 24 108 320 750 1512 2744 4608
```

As remarked in GKP, If `g` is the function such that `g k` yields the Fibonacci number whose index is `k`, then `g` must satisfy the following relation:

```
g = g@(-&1) + g@(-&2) + =&1
```

assuming that the result of g on negative integers is zero.

The corresponding relation for the generating function is given by any of the following :

```
(g G z) = (z * g G z) +((z^2) * g G z) + z
((g G z) * (1 + (-z) + (-z^2))) = z
(g G z) = z % (1 + (-z) +(-z^2))
(g G z) = (0 1 p. z) % (1 _1 _1 p. z)
(g G z) = (0 1 p. z) * %(1 _1 _1 p. z)
```

D. CONVOLUTIONS

The function pp used for the product of polynomials in §2E is a simple example of a convolution. For example:

```
pp=: +//.@(*/)
c=: 1 2 1 [ d=: 1 3 3 1
,. c(pp ; */ ; </.@(*/)) d
```

1 5 10 10 5 1
1 3 3 1 2 6 6 2 1 3 3 1
1
3 2
3 6 1
1 6 3
2 3
1

We may also use functions such as Fibonacci in a convolution such as that of GKP7.60:

```
   F=: 0&~:`(+/@:F@(<:@<: , <:))@.(1&<)"0
   F i. 15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
   L=: # {. +//.@(F */ F)
   R=: 1r5"0 * (2: * ] *F@>:) - (>: * F)
   ((<@L\),.(<@R\)) x=: i. 10
```

0	0
0 0	0 0
0 0 1	0 0 1
0 0 1 2	0 0 1 2
0 0 1 2 5	0 0 1 2 5
0 0 1 2 5 10	0 0 1 2 5 10
0 0 1 2 5 10 20	0 0 1 2 5 10 20
0 0 1 2 5 10 20 38	0 0 1 2 5 10 20 38
0 0 1 2 5 10 20 38 71	0 0 1 2 5 10 20 38 71

```
│0 0 1 2 5 10 20 38 71 130│0 0 1 2 5 10 20 38 71 130│
└─────────────────────────┴─────────────────────────┘
```

E. EXPONENTIAL GENERATING FUNCTIONS

As shown in §B, the function `gex=: ^ t.` is a generating function for the exponential. For example:

```
   gex=: ^ t.
   ]ce=: gex i=: i. 8
1 1 0.5 0.1666667 0.04166667 0.008333333 0.001388889 0.0001984127
   ce p. z=: 0.1*i.11
1 1.10517 1.2214 1.34986 1.49182 1.64872 1.82212 2.01375 2.22554
   2.45959 2.71825
   ^z
1 1.10517 1.2214 1.34986 1.49182 1.64872 1.82212 2.01375 2.22554
   2.4596 2.71828
```

We may also replace the polynomial `p.` by the equivalent sum over powers, or by a sum over some other function, using suitably modified coefficients. For example:

```
   p=: +/@([ * (] ^ exp)) " 1 0
      exp=: i.@#@[
   q=: +/@([ * (] ^ exp) % !@exp) " 1 0
   hex=: ! * ^ t.
   he=: hex i
   ce p z
1 1.10517 1.2214 1.34986 1.49182 1.64872 1.82212 2.01375 2.22554
   2.45959 2.71825
   he q z
1 1.10517 1.2214 1.34986 1.49182 1.64872 1.82212 2.01375 2.22554
   2.45959 2.71825
```

The generating function `hex` has a particularly simple form that also appears in the case of related functions such as the sine and cosine, and is therefore called an *exponential* generating function. For example:

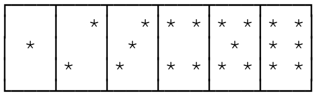
```
   hex i
1 1 1 1 1 1 1 1
   hsine=: ! * 1&o. t.
   hcosine=: ! * 2&o. t.
   (hex , hsine ,: hcosine) i
1 1  1  1 1 1  1  1
0 1  0 _1 0 1  0 _1
1 0 _1  0 1 0 _1  0
```

Chapter 8
DISCRETE PROBABILITY

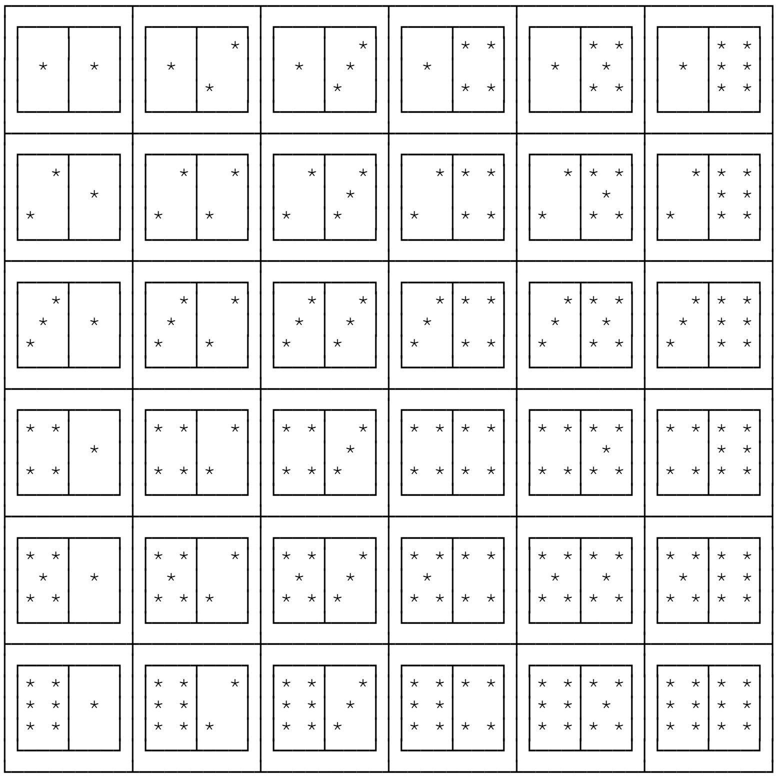
A. DEFINITIONS

To illustrate a *probability space*, GKP defines graphic displays of a set of dice (D), and the set of all possible pairs of dice (D^2):

```
   ]D=: <"2 ' *'{~r=: 3 3$"1 #: 16 68 84 325 341 365
```

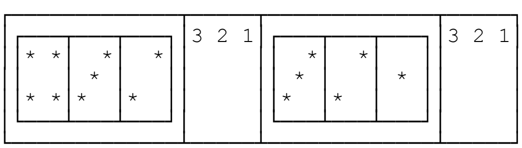


```
   ]D2=: { ,&<~ D
```



Although graphic displays may aid visualization, it is more convenient to represent the elements of a discrete space by integers (such as `1 2 3 4 5 6` for the number of pips on a die, or by `i.6`), and define functions to convert to and from graphic displays. Thus:

```
   di=: (d=: {&D) ^:_1
   d1i=: (d1=: d@<:)^:_1
   (d 3 2 1);(di d 3 2 1);(d1 3 2 1);(d1i d1 3 2 1)
```




```
   next=: 6&|@>:
   NEXT=: next&.di
   next i. 6
1 2 3 4 5 0
```

```
   NEXT D
┌─────┬─────┬─────┬─────┬─────┬─────┐
│   * │   * │ * * │ * * │ * * │     │
│     │  *  │     │  *  │ * * │  *  │
│ *   │ *   │ * * │ * * │ * * │     │
└─────┴─────┴─────┴─────┴─────┴─────┘
   (D =/ NEXT D) ; (i =/ next i=: i. 6)
┌───────────┬───────────┐
│0 0 0 0 0 1│0 0 0 0 0 1│
│1 0 0 0 0 0│1 0 0 0 0 0│
│0 1 0 0 0 0│0 1 0 0 0 0│
│0 0 1 0 0 0│0 0 1 0 0 0│
│0 0 0 1 0 0│0 0 0 1 0 0│
│0 0 0 0 1 0│0 0 0 0 1 0│
└───────────┴───────────┘
```

The (pseudo-) random number generator `?` can be used to experiment with probabilities. For example:

```
   count=: +/@(sums =/ pos@[)              Count of possible outcomes
     sums=: +/@>:@?@(,$6:)                 Sums over groups of k dice
     pos=: ] }. Ei@(6: * ])                Possible outcomes for k dice
   pos 2
2 3 4 5 6 7 8 9 10 11 12
   2 count 10000
272 562 815 1125 1390 1696 1336 1066 861 577 300
   theoretical=: 1 2 3 4 5 6 5 4 3 2 1 % 36
   <. 1r2 + 10000 * theoretical
278 556 833 1111 1389 1667 1389 1111 833 556 278
```

The mean, median, and mode are defined as in GKP, except that when two elements compete for the median (in the case of an even number of elements) or several compete for the mode, the result is their mean:

```
   mean=: +/ % #
   median=: [: mean (<.,>.)@-:@<:@# { /:~
   w=: 1 2 7 4 1 7 2 1 7 5 0 6
   /:~ w
0 1 1 1 2 2 4 5 6 7 7 7
   median w
3
   mode=: mean@((= >./)@:({."1) # {:"1)@freq
     freq=: (# , {.)/.~
   y=: 2 3 2 4 3 5
   freq y
2 2
2 3
1 4
1 5
   ~. y
2 3 4 5
   mode y
2.5
   (mean,median,mode) 3 1 4 1 5
2.8 3 1
```

B. MEAN AND VARIANCE

The variance and standard deviation may be defined and used as follows:

```
   var=: mean@*:@(] - mean)
   sd=: %:@var
```

```
   (var , 35r12"_ , sd) 1 2 3 4 5 6
2.91667 2.91667 1.70783
```

The *sample variance* is defined as follows:

```
   Svar=: <:@# %~ +/@*:@(] - mean)
   Svar 1 2 3 4 5 6
3.5
   6r5*var 1 2 3 4 5 6
3.5
```

The foregoing definition of the sample variance may be compared with the definition given in GKP8.20:

```
   VX=: (+/@*: % <:@#) - *:@(+/)% # * <:@#
   (VX , Svar) x=: 7 11 8 5 4 6 10 8 8 7
4.48889 4.48889
```

GPK8.22 returns to the question of the number of stationary points in a permutation, for which we defined in §5E the functions:

```
   DNSP=: +/@(NSP =/ Ei@{:@$)
     NSP=: +/@(1: = #@>@C.)"1
   ]r=: DNSP ?~ 10000#5
3636 3801 1640 836 0 87
   r%+/r
0.3636 0.3801 0.164 0.0836 0 0.0087
```

These "experimental" results may be compared with the theoretical results obtained by normalizing the penultimate row of the final table of §5E:

```
   s=: 44 45 20 10 0 1
   s%+/s
0.3666667 0.375 0.1666667 0.08333333 0 0.008333333
```

To experiment with the mean and variance of various functions on various domains, we define the following adverbs:

```
   M=: mean@:
   V=: var@:
   ]a=: 1+i.8
1 2 3 4 5 6 7 8
   (*:M,*:V) a
25.5 446.25
   3 2 $ (*:M , *:V , %M , %V , ]M , ]V) a
     25.5     446.25
0.3397321 0.07550983
      4.5       5.25
```

They may also be applied to the function `NSP` to yield the results stated in GKP; that is, `1` for both mean and variance when applied to the set of all permutations of a given order, and approximations to these results for a random set:

```
   (NSP M , NSP V) ?~ 1000#4
1.079 1.10676
   allp=: i.@! A. i.
   (] ; ,.@:NSP ; NSP M ; NSP V) allp 3
┌─────┬─┬─┬─┐
│0 1 2│3│1│1│
│0 2 1│1│ │ │
│1 0 2│1│ │ │
│1 2 0│0│ │ │
│2 0 1│0│ │ │
│2 1 0│1│ │ │
└─────┴─┴─┴─┘
```

C. PROBABILITY GENERATING FUNCTIONS

If `a` is a non-negative integer list that sums to `1`, then `a&PR` defines a (discrete) *probability distribution*, where `PR` is defined as follows:

```
   PR=: {~ :: 0:"1 0              Result is 0 if indexing of the list fails
   U=: 0:, ] %~ ] # 1:            List for uniform distribution in 1 to argument
   c10=: U 10
   c10
0 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
   p10=: c10&PR                   Uniform distribution function for 1 to 10
   (p10 0 1 2 11 12) ; (+/p10 2 4 6 8 10)
┌───────────────┬───┐
│0 0.1 0.1 0 0  │0.5│
└───────────────┴───┘
   c6=: U 6
   p6=: c6&PR                     Uniform distribution for 1 to 6 (a fair die)
   p6 0 2 4 6 8 10
0 0.1666667 0.1666667 0.1666667 0 0
```

The generating function for the distribution `d&PR` is then `G=: d&p.`. Thus:

```
   g6=: c6&p.
   g6 i. 8
0 1 21 182 910 3255 9331 22876
```

The function `f t. @ i.` applied to `n` yields the first `n` terms of the Taylor series for `f`, and `(f t.@i. n)&p.` is therefore the `n`-term approximation to `f`. For example:

```
   e7=: ^t.@i. 7
   ,. e7 ; (e7&p. x) ; (^ x=: i.4)
┌──────────────────────────────────────────────────────────┐
│1 1 0.5 0.1666667 0.04166667 0.008333333 0.001388889      │
├──────────────────────────────────────────────────────────┤
│1 2.71806 7.35556 19.4125                                 │
├──────────────────────────────────────────────────────────┤
│1 2.71828 7.38906 20.0855                                 │
└──────────────────────────────────────────────────────────┘
```

We will define an adverb `tc` such that `f tc m,n,s` tests and yields the first Taylor coefficients of `f`, beginning with a minimum number `m` of them, and continuing until the number reaches `n`, or their sum reaches `s`:

```
   tc=: 1 : '] }. (, x.t.@#)@]^:test^:_ x.t.@i.@{.'
     test=: ({.@[ > #@]) *. ({:@[ > +/@])
   ,. (^ tc 1 _ 2.718) ; (^ tc 1 3 2.718) ; (^ tc 5 3 _)
┌──────────────────────────────────────────────────────────┐
│1 1 0.5 0.1666667 0.04166667 0.008333333 0.001388889      │
├──────────────────────────────────────────────────────────┤
│1 1 0.5                                                   │
├──────────────────────────────────────────────────────────┤
│1 1 0.5 0.1666667 0.04166667                              │
└──────────────────────────────────────────────────────────┘
```

It will often be more convenient to use the corresponding conjunction `TC`, defined as follows:

```
   TC=: 2 : 'x. tc y.'
   ^ TC 1 _ 2.7
1 1 0.5 0.1666667 0.04166667
```

If `f` is the generating function for a probability distribution, then its Taylor coefficients must be non-negative, and must sum to `1`. We will exploit these facts to define an adverb `g` such that the expression `f g` yields all significant Taylor coefficients of `f`:

```
   g=: TC 1 _ 1
```

```
   g6 g
0 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
```

As a companion to `g` we will define the adverb `G` as follows:

```
   G=: &p.
   f=: c6 G
   f i. 8
0 1 21 182 910 3255 9331 22876
   q=: g6 g G
   q i. 8
0 1 21 182 910 3255 9331 22876
```

The adverbs `tc` and `g` apply only to functions in the domain of the Taylor coefficients adverb `t.`; others may be treated by the expression `(f r) %. r ^/ i. n`, which yields the coefficients of the `n`-term polynomial that best fits `f` on the range of arguments `r`. For example:

```
   r=: 1+i. 20 [ n=: 10
   log=: 10&^.
   4 5$log r
        0   0.30103 0.4771213   0.60206 0.69897
0.7781513 0.845098   0.90309 0.9542425       1
  1.04139  1.07918   1.11394   1.14613 1.17609
  1.20412  1.23045   1.25527   1.27875 1.30103
   2 5 $ c=: (log r) %. r ^/ i. n
  _0.5689842   0.7840836 _0.2696426  0.06350159 _0.009729294
0.0009699834 _6.22926e_5 2.48047e_6 _5.56407e_8  5.36916e_10
   4 5 $ c p. r
0.0001392442 0.3003129 0.4783184 0.6017181 0.6982362
    0.778159 0.8456566   0.90346 0.9540612 0.9995265
     1.04114   1.07939   1.11437    1.1463    1.1758
     1.20374   1.23057   1.25575   1.27837   1.30111
```

The function `g6` is the generating function for the probability distribution of a die and, as stated in GKP, the function `g6*g6` is the generating function for the sum of two dice. Thus:

```
   g6 x=: i. 8
0 1 21 182 910 3255 9331 22876
   (g6*g6) x
0 1 441 33124 828100 1.0595e7 8.70676e7 5.23311e8
```

As stated in GKP8.28, the first derivative of these functions evaluated at `1` gives the means of the distributions:

```
   ((g6 D.1) , (g6*g6)D.1) 1
3.5 7
```

We may test these results by evaluating the means directly as follows:

```
   mean=: +/ % #
   mean d=: 1 2 3 4 5 6
3.5
   ]q=: { 2 # <d
```

1 1	1 2	1 3	1 4	1 5	1 6
2 1	2 2	2 3	2 4	2 5	2 6
3 1	3 2	3 3	3 4	3 5	3 6
4 1	4 2	4 3	4 4	4 5	4 6
5 1	5 2	5 3	5 4	5 5	5 6
6 1	6 2	6 3	6 4	6 5	6 6

```
   ]r=: +/&> q
2 3 4  5  6  7
3 4 5  6  7  8
4 5 6  7  8  9
5 6 7  8  9 10
6 7 8  9 10 11
7 8 9 10 11 12
   mean ,r
7
```

Similarly for the sum of three dice:

```
   ((g6*g6*g6)D.1 (1)) , mean ,+/&>{3#<d
10.5 10.5
```

GKP8.31 gives the following expression for the variance for two dice:

```
   h2=: g6*g6
   (h2 D.2 + h2 D.1 - *:@(h2 D.1)) 1
5.83333
```

This result agrees with the following direct calculation of the mean of the squares of the differences from the mean:

```
   mean *: , r - mean , r
5.83333
```

GKP8.33 uses notation of the form f(1+t) to denote the function `f@>:`, and the coefficients of powers of t in the theorem are therefore obtained by applying the conjunction `tc` to it. In the case of the function `h2=: g6*g6` (the generating function for the sums of a pair of dice), this becomes:

```
   h2=: g6*g6
   h2@>: tc 1 5 _
1 7 23.9167 52.5 81.8611
```

According to GKP8.33, these results should agree with:

```
   (!i.5) %~ (h2,(h2 D.1),(h2 D.2),(h2 D.3),(h2 D.4))1
1 7 23.9167 52.5 81.8611
```

Similarly, the left limb of GKP8.41 (which introduces *cumulants*) is expressed as `h2@^`, and the first eight coefficients of the right limb are obtained as follows:

```
   h2@^ tc 1 8 _
1 7 27.4167 77.5833 174.854 330.274 539.393 777.838
```

D. FLIPPING COINS

We will define the function `q` as the complement of the function `p`, and begin by defining `p` to be the *fair* probability for the toss of a coin:

```
   q=: 1: - p
   p=: 1r2"0
```

The binomial distribution `bd` is then defined such that `k bd n` gives the probability of `k` heads in `n` tosses:

```
   bd=: ! * (p ^ [) * (q ^ -~)
   bd"0/~i. 6
1 0.5 0.25 0.125 0.0625 0.03125
0 0.5  0.5 0.375   0.25 0.15625
0   0 0.25 0.375  0.375  0.3125
0   0    0 0.125   0.25  0.3125
0   0    0     0 0.0625 0.15625
0   0    0     0      0 0.03125
```

To explore probabilities for a biased coin we will define an adverb as follows:

```
   B=: 1 : '! * (x."0 ^ [) * ((1: - x."0) ^ -~)'
   T=: ("0) / ~
   (1r2 B T ; 1r4 B T ; 0 B T ; 1 B T)i. 4
┌──────────────────┬──────────────────────────────┬───────┬───────┐
│1 0.5 0.25 0.125  │1 0.75 0.5625 0.421875        │1 1 1 1│1 0 0 0│
│0 0.5  0.5 0.375  │0 0.25  0.375 0.421875        │0 0 0 0│0 1 0 0│
│0   0 0.25 0.375  │0    0 0.0625 0.140625        │0 0 0 0│0 0 1 0│
│0   0    0 0.125  │0    0      0 0.015625        │0 0 0 0│0 0 0 1│
└──────────────────┴──────────────────────────────┴───────┴───────┘
```

The generating function for the probability that the first head shows up at the `k`th toss is given by GKP8.58 as pz over 1-qz. For a fair coin, this is the rational function `(c&p.)%(d&p.)`, where `c` and `d` are as shown below. We will use the conjunction `over` from §5C as follows:

```
   c=: 0 1r2
   d=: 1 _1r2
   over=: ([.&p.) % (].&p.)
   c over d
0 0.5&p. % 1 _0.5&p.
   2 5 $ ser=: c over d t. i. 10
      0      0.5      0.25      0.125       0.0625
0.03125 0.015625 0.0078125 0.00390625 0.001953125
```

The probability of exactly `n` heads is given the `n`th power of `c over d`, and its generating function is therefore given by `cn over dn t.`, where `cn` and `dn` are the `n`th powers of the coefficients `c` and `d`. For example:

```
   pp=: +//.@(*/)
   c5=: c&pp^:5 (1)
   d5=: d&pp^:5 (1)
   c5,:d5
0    0   0     0      0  0.03125
1 _2.5 2.5 _1.25 0.3125 _0.03125
   3 5 $ c5 over d5 t. i. 15
        0         0          0         0          0
  0.03125  0.078125  0.1171875 0.1367188  0.1367188
0.1230469 0.1025391 0.08056641 0.0604248 0.04364014
```

The zeros in the first row show that (as expected) there is no probability of five heads for the cases of 0-4 throws; the first element of the next row is the expected `1r2^5`.

For the polynomials that occur in further theorems in this section of GKP it may be convenient to use functions for both the sums and products of polynomials. For example the coefficients for the numerator and the denominator of GKP8.69 [$p^2q^3z^5$ and $p^2q^3z^5+(1+pq^2z^3)(1-z)$] may be obtained as follows:

```
   ps=: +/@,:
   num=: 0 0 0 0 0 1r32
   den=: num ps 1 0 0 1r8 pp 1 _1
   3 5 $ num over den t. i. 15
         0          0          0          0          0
   0.03125    0.03125    0.03125 0.02734375 0.02734375
0.02636719 0.02587891 0.02490234 0.02416992 0.02337646
```

E. HASHING

Among the probability generating functions discussed in this section of GKP, Theorem 8.92 introduces a new problem, the application of a polynomial with coefficients `s` (representing *search* probabilities) to the results of (atop) a second polynomial with coefficients `((m-1),1)%m` (where `1%m` is the probability of success). For this we will use the conjunction `atop` defined in §5B in an example as follows:

```
   ]s=: 6#1r6
666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
   ]t=: ((m-1),1)%m =: 0.1
_9 10
   s atop t
_8857.33 50191.7 _113833 129167 _73333.3 16666.7
   0 1 pp s atop t
0 _8857.33 50191.7 _113833 129167 _73333.3 16666.7
```

F. NOTATION

Catalogue and Cartesian product. The phrase `{b` produces boxed lists comprising one item from each of the boxes of `b`. The special case of two boxes is analogous to the Cartesian product. For example:

```
   ]b=: 'AB';'012';'abcd'
```

AB	012	abcd

```
   {b
```

A0a	A0b	A0c	A0d
A1a	A1b	A1c	A1d
A2a	A2b	A2c	A2d

B0a	B0b	B0c	B0d
B1a	B1b	B1c	B1d
B2a	B2b	B2c	B2d

Chapter 9
ASYMPTOTICS

A. O NOTATION

GKP defines a sum function and two approximations to it as follows:

```
   S=: +/@(Ei ! 3&*)"0
   S1=: (2: * ] ! 3&*)"0
   S2=: ((2: - 4: % ]) * ] ! 3&*)"0
   (S,S1,:S2) (i. 8),100
 1  4 22 130 794    4944 31180 198440 8.16329e81
 2  6 30 168 990    6006 37128 232560  8.3165e81
__ _6  0  56 495 3603.6 24752 166114 8.15017e81
```

GKP9.1 adds the term O with argument $1/n^2$ to the factor `2:-4:%]` to indicate that the approximation `S2` gives a "relative error of order $1/n^2$". The function O (or "O notation") suggests that the difference `|@(S-S2)` is dominated by the function $1/n^2$ (that is, `^&_2`):

```
   (|@(S-S2) , ^&_2 ,: !@(S-S2) <: ^&_2) (i. 8),100
_ 10   22         74    299 1340.4       6428    32325.7 1.31217e79
_  1 0.25 0.1111111 0.0625   0.04 0.02777778 0.02040816     0.0001
1  0    0         0      0      0          0          0          0
```

The zeros in the last row show that this is not entirely true. As defined in §9.2 of GKP, the O notation implies only that the dominance is *asymptotic*, true for arguments that exceed some specified constant lower limit `lim`, and for multiplication of the dominating function by some constant `c`.

The dominance asserted by the O notation may therefore be characterized by the expression `g >: c&*@(^&n)` *or*ed with the condition that the argument exceeds some specified lower limit. Thus:

```
   <&lowerlim +. g >: c&*@(^&n)
```

In other words, the result is true (`1`) if either the argument is less than the specified limit *or* the function `g` (the difference `S-S2` in the foregoing example) is greater than or equal to the specified constant `c` times the argument raised to the specified power `n`.

We now define a conjunction `O` such that the expression `g O (lim,c,n)` yields a tautology if `g` is indeed asymptotic to zero for the specified parameter values:

```
   O=: 2 : '<&({. y.) +. |@x. <: (1&{ y.)&*@(^&({: y.))'
   t=:(S-S2) O 50 4 _2
   t
<&50 +. |@(S - S2) <: 4&*@(^&_2)
   t 0 1 2 3 4 5 100
1 1 1 1 1 1 0
```

The approximation to the harmonic numbers given in GKP9.28 is, as stated, of order $1/n^6$

```
   H=: +/@%@>:@i."0
   HA=: (^. + 0.5772156649"_ + +/@(^&_1 _2 _4 % 2 _12 120"_))"0
   (H , HA , |@(H-HA) ,: ^&_6) 1+i. 6
          1        1.5     1.83333      2.08333    2.28333       2.45
    1.00222    1.50005     1.83334      2.08333    2.28333       2.45
0.002215665 5.03455e_5   4.9083e_6   9.11437e_7 2.44001e_7 8.26877e_8
          1   0.015625 0.001371742 0.0002441406     6.4e_5 2.14335e_5
   (H-HA) O 1 1 _6 i.20           The conjunction O could be mistaken for a zero
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   (H-HA) O 1 1 _8 i.20           The approximation is not of order _8
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
```

In dealing with asymptotics we must, as remarked by GKP, "THINK BIG, when imagining a variable that approaches infinity". Consequently, there are few common asymptotic approximations that can be used to illustrate the conjunction `O` defined here, at least when run on an underlying computer system that limits the precision to about sixteen decimal digits. However, the functions that it generates give an alternative view of the definition of the O notation; in particular, one that brings together (in a difference function) the two functions concerned. This helps to make clear the question of which of the several functions denoted by Sn the function in GKP9.1 approximates to order $1/n^2$:

```
   S=: +/@(Ei ! 3&*)"0
   S1=: (2: * ] ! 3&*)"0
   S2=: ((2: - 4: % ]) * ] ! 3&*)"0
   (S-S2) O 50 4 _2
<&50 +. |@(S - S2) <: 4&*@(^&_2)
```

Experimentation with further expressions in this chapter will require first translating them into J. Such translation can be simplified by recognizing the occurrence of polynomials (commonly in the reciprocal of the argument n). For example, the approximation `HA` defined above may be written as `c&ha`, where:

```
   ha=: ^.@] + [ p. %@]
   c=: 0.5772156649 1r2 _1r12 1r120
   (c&ha ,. HA) i.6
      0 _1.03367e308
1.00222      1.00222
1.50057      1.50005
1.83354      1.83334
2.08343      2.08333
2.28339      2.28333
```

Similarly, the right limb of GKP9.29 (excluding the term in O) may be defined as follows:

```
   d=: 1 1r12 1r288 _139r51840
   GKP9_29=: (%:@(2p1&*@]) * %&1x1@] ^ ]) * [ p. %@]
   (! ,. d&GKP9_29 ,. ! - d&GKP9_29) i. 8
   1         0           1
   1 0.9997111  0.0002889268
   2   1.99999    1.45163e_5
   6         6     1.4362e_6
  24        24   _3.47278e_6
 120       120   _1.40573e_5
 720       720   _5.45381e_5
5040      5040 _0.0002442346
```

The phrase `%@]` that defines the argument to the polynomial of GKP9.29 may also be expressed as the power `^&_1@]`, and other powers of the argument may also be required. For example, the approximation to $1/n^2+2$ (page 444 of GKP) requires the reciprocal square:

```
   ]e=: 0,_2^i.4
0 1 _2 4 _8
   F444=: [ p. ^&_2@]
   G=: %@(0 1 1&p.)                 1/n2+2 as a polynomial
   (G,.e&F444) I.8

_       __
      0.5         _5
 0.1666667    0.15625
0.08333333  0.0906874
      0.05 0.05554199
0.03333333 0.03703552
0.02380952 0.02631554
0.01785714 0.01960779
```

REFERENCES

1. Graham R.L., D.E. Knuth , and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley, 1988

2. Iverson, K.E. *J Introduction and Dictionary*, Iverson Software, 1995

3. " *Arithmetic* " , 1992

4. " *Calculus* " , 1992

5. McDonnell, E.E. *Complex floor*, APL73 Conference Proceedings, ACM

6. Hui, R.K.W and K.E. Iverson, *Representations of Recursion*, APL95 Conference Proceedings, ACM.

7. Abramowitz, M. and I. Stegun, *Handbook of Mathematical Functions*, U.S. Department of Commerce, National Bureau of Standards, Applied Mathematics Series #55, Tenth Printing, 1972.

ACKNOWLEDGMENT

I am indebted to Roger Hui for numerous suggestions arising from his careful review of a draft of this book, as well as for timely implementation of new facilities such as the functions `p:` and `q:` for the treatment of primes and prime factorization.

INDEX