C Exercise Solutions

C.1 Chapter 1

C.1.* Age

Write your age as an int, a short, a long and a byte.

```
42
42h
42j
0xa6
```

C.1.* Truth

Write a boolean that indicates whether the answers you gave for the previous exercise represent your true age.

```
0b
```

C.1.* Temperature

Write the normal body temperature (your choice of F or C) as a float and as a real in normal and scientific notation.

```
98.2
98.2e
9.82e1
9.82e1e

38.6
38.6e
8.86e1
3.86e1e
```

C.1.* Name

Write your first (given) name as a symbol

```
`jeffry
```

C.1.* Birthday

Write your birthday as a date and assign it to a variable `bday`

```
bday:1942.07.04
```

C.1.* Birthday Constituents

Write expressions that extract the year, month and day from the variable `bday`.

```
bday.year
bday.mm
bday.dd
```

C.1.* Time

Write midnight and one second before midnight as times with milliseconds.

```
00:00:00.000
23:59:59.000
```

C.1.* Santa's Arrival

Assume Santa leaves the North Pole at precisely the stroke of midnight on Christmas eve and it takes him exactly one hour, two minutes, three seconds and four milliseconds to arrive at your chimney. Write his arrival time for this Christmas as a datetime.

```
2007.12.25T01:02:03.004
```

C.1.* Date Number

Express noon of January 19th, 2000 as a float.

```
18.5
```

C.1.* Character Nulls

Write two expressions that assign the symbol null and the character null to variables whose names are your first and last names.

```
zaphod:`
beeblebrox:" "
```

C.2 Chapter 2

C.2.* Birthday List

Write a simple list whose items are ints with the month, day and year of your birthday. Write this same list as a general list and assign it to the variable `b`. Index the day of your birthday from `b`. Write an expression that finds the position of your special year in `b`.

```
7 4 1942
b:(7;4;1942)
b[1]
```

```
b?1942
```

C.2.* Initials

Write your initials as a simple list of symbols. Do the same as a list of char.

```
`j`a`b
"jab"
```

C.2.* Your Info

Assign a general list `L` composed of two lists. The first item is the birthday list from §2.w.* and the second is the symbol initials list from §2.w.*. Index the day of your birthday from `L` with repeated indexing and indexing at depth.

```
L:(b;`j`a`b)
L[0][1]
L[0;1]
```

C.2.* Boolean List

Write a boolean simple list whose items indicate whether each component of your birthday list in §2.w.* is even or odd and assign it to `eo`. Write an expression that extracts only the first and last items of `eo`.

```
eo:011b
eo[0 2]
```

C.2.* Empty Lists

Write an expression for a list that contains the following two items: the empty list and a list containing the empty list. Write an int list containing the count of the items of the previous list.

```
(();enlist())
0 1
```

C.2.* List Notation

We find,

```
1 -2 3[1]
-2
`a`b`c 2
`c
(();())[0]
_
(())0
_
```

The last two results are the empty list.

C.2.* Matrices

Create a variable `m` that is a 4 by 3 matrix whose entries represent the number of days in each month in a non-leap year. Write an expression that extracts the number of days in the second month of the third quarter. (***Hint***: remember relative to zero). Write expressions that retrieve the number of days in all the months of the third quarter and in the first month of each quarter.

Pluto is annoyed at being demoted to a planetoid, so it leaves its orbit and makes a pass at Earth. All the nations of Earth unite to fire their entire stockpile of nuclear weapons and missiles at the angry snowball, reducing it to planetessimals but not avoiding the extinction of all "intelligent" life. The near miss has severely destabilized the Earth's motion. As the Royal Astronomer dolphin, write an expression that assigns 42 days to the last month of the first and last quarters.

```
m:(31 28 30;30 31 30;31 31 30;31 30 31)
m[2;1]
m[2;]
m[;0]

m[0 3;2]:42
```

C.2.* Flipping Out

We find,

```
flip til 5
'type
flip enlist til 5
0
1
2
3
4
```

The first expression attempts to transpose a vector, which is not a rectangular list because it is a list of atoms rather than a list of lists of the same size. The second expression transposes a 1x5 list to a 5x1 list.

C.3 Chapter 3

C.3.* No Stinking Parentheses

Write an expression that returns the value of two-thirds x plus one without using parentheses.

```
1+2*x%3
```

C.3.* Minus

What is the value of,

```
4--4--4
4
```

C.3.* Vector Operation

Using a single operator and no parentheses, write an expression that returns a boolean list showing whether each month has 30 days.

```
30=31 28 30 30 31 30 31 31 30 31 30 31
```

C.3.* Match vs. Equals

Two numeric atoms both have the value 42 and are equal but they do not match. How can this happen? Give an example.

The types are different:

```
42=42h
1b
42~42h
0b
```

C.3.* Equals…pause…Not

Write the test `a<>b` without using <>. Write the test `a>=b` without using >= or `|`.

```
not a=b
not a<b
```

C.3.* Sign

Write an expression that returns the sign of a numeric `x` without using `signum`.

```
(x>0)-x<0
```

C.3.* Division

What is the result of

4 / 2

(Note spaces) ? Are the results of `4%2` and `4.0%2.0` the same? Write an expression that performs this test.

```
4
```
Yes: `(4%2)~4.0%2.0`

C.3.* Power

Write an expression that returns 2 raised to the 5th power without using `xexp`.

```
2*2*2*2*2
```

-or-

```
exp 5*log 2
```

C.3.* Dates

Write a comparison that tests whether your birthday is in the second half of your birth year. Write an expression that computes the number of days between your birthday and Christmas of your birth year. Write an expression that determines the day 42 days after your birthday and assigns it to the variable b42. Write an expression that returns the earlier of b42 and New Years day of the year following your birth year.

```
1942.07.04>=1942.07.01
1942.12.25-1942.07.04
b42:1942.07.04+42
b42&1943.01.01
```

C.3.* Date Calculation

Write the simplest expression that returns the quarter (relative to 0) of an arbitrary date `d`.

```
floor d.mm%4
```

C.3.* Adding Time

Write an expression that adds four hours to the time `12:34:56.789`

Without using cast,

```
12:34:56.789+4*60*60*1000
16:34:56.789
```

Using cast,

```
12:34:56.789+`int$04:00:00.000
16:34:56.789
```

-or-

```
`time$12:34:56.789+04:00:00.000
16:34:56.789
```

C.3.* Adding Time

Write an expression that adds four hours to the time 12:34:56.789

Without using cast,

```
12:34:56.789+4*60*60*1000
16:34:56.789
```

Using cast,

```
12:34:56.789+`int$04:00:00.000
16:34:56.789
```

-or-

```
`time$12:34:56.789+04:00:00.000
16:34:56.789
```

3.w.* Round About Midnight

Write an expression that expresses the difference between the datetimes below as milliseconds.

```
z1:2000.01.01T23:59:59.999
z2:2000.01.02T00:00:00.001
```

Without cast,

```
floor 24*60*60*1000*z2-z1
2
```

With cast

```
`time$`datetime$z2-z1
00:00:00.002
```

C.4 Chapter 4

C.4.* Average

Write the function that returns the average of two arguments a and b without using parentheses.

Do the same for arguments x and y. Same for y and z.

```
avg1:{[a;b].5*a+b}
avg2:{.5*x+y} / can use default positional parms
avg3:{[y;z] .5*y+z} / cannot use default positional parms
```

C.4.* Locals and Globals

A new q session is started and the following lines are typed into the console,

```
a:42
b:98.6
f:{a::4;b:7;c::9;b::27;a+b+c+x}
f 2
```

What is displayed on the q console after the last entered line? What global variables exist in the workspace and what are their values?

```
f 2
42
a
4
b
98.6
c
9
```

C.4.* Amend

What is the result of the following function valuation?

```
{y|z&:y*:x+:1}[6;6;43]
42
{y|z|:y*:x+:1}[6;6;43]
43
```

C.4.* Projection

Given the list,

```
L:(1 2;3 4 5;10 20 30)
```

we find,

```
L[;0]+:100
L
(101 2;103 4 5;110 20 30)
```

C.4.* Weighted Vector

Write a function that takes a numeric vector and weights each item by its ordinal (i.e., first, second, etc) divided by the number of items. For performance, use a single division.

```
wt:{(1%n)*x*1+til n:count x}
```

C.4.* Metrics

Write an expression that takes a numeric list `L` and returns a vector containing the minimum, maximum and average of the items in `L:10?100`.

```
(min;max;avg)@\:L
6
93
51.3
```

C.4.* Triangles

We find,

```
1+til each 1+til 5
,1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

(1+til 5) #' 1+til 5
,1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

C.4.* Squares

We find,

```
show (til 5) +/: til 5
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
show (til 5) |/: til 5
0 1 2 3 4
1 1 2 3 4
```

```
2 2 2 3 4
3 3 3 3 4
4 4 4 4 4
show (reverse til 5) |/: reverse til 5
4 4 4 4 4
4 3 3 3 3
4 3 2 2 2
4 3 2 1 1
4 3 2 1 0
```

C.4.* Diagonals

We find,

```
(til 5) =/: til 5
10000b
01000b
00100b
00010b
00001b
(reverse til 5) =/: til 5
00001b
00010b
00100b
01000b
10000b
(til 5) {x*x=y}/: til 5
0 0 0 0 0
0 1 0 0 0
0 0 2 0 0
0 0 0 3 0
0 0 0 0 4
```

C.4.* Each

Write a triadic function `wrap` whose arguments are a string, a list of strings and a string, which prepends the first argument and appends the third argument to each item of the second argument.

```
wrap:{x,/:y,\:z}

L:("s1";"s2";"s3")
wrap["<";L;">"]
("<s1>";"<s2>";"<s3>")
```

C.4.* Verb Amend

Given,

```
L:("abc";"de";"xyz")
```

we find,

```
@[L;2 1;&;"m"]
("abc";"de";"mmm")
.[L;2 1;&;"m"]
("abc";"de";"xmz")
```

C.4.* Operating on a List of Pairs and a Pair of Lists

Given the lists,

```
L1:(1 10;2 20;3 30)
L2:(1 2 3;10 20 30)
```

and the function,

```
f:{x+y}
```

we find,

```
f .' L1
11 22 33

(f') . L2
11 22 33
```

C.4.w* Monotone Sequences

Write an expression that returns `1b` if the items of a numeric list are monotone decreasing—i.e., successive values decrease—and `0b` otherwise. Same for strictly monotone—i.e., no equal values.

We find,

```
all 0w<=': 10 9 8 8 6
1b
all 0w<=': 10 9 8 8 6 42
0b

all 0w<': 10 9 8 8 6
0b
all 0w<': 10 9 8 7 6
1b
```

Write an expression that returns `1b` if a boolean vector has leading ones and trailing zeroes and

`0b` otherwise. The degenerate case of all `0b` should also pass.

The same expression as above works since leading ones correspond to monotone decreasing booleans.

```
all 0w<=': 1110b
1b
all 0w<=': 11101b
0b
all 0w<=': 0000b
1b
```

C.5 Chapter 5

C.5.* Type

Given,

```
L:(1 2;`3`4;"5 6")
```

we find,

```
type L
0h
type each L
6 11 10h
```

C.5.* Cast

We find,

```
(1h;"h";`datetime)$98.6
(1b;99h;2000.04.08T14:24:00.000)

"i"$"F"$string 42.2
42
```

C.5.* Empty Lists

We find,

```
(())~()
1b
(`$())~()
0b
((),())
```

```
()
```

C.5.* Ticker Enumeration

Given the list of ticker symbols,

```
sym:`ibm`msft`ge`msft`ge`ibm
```

create an enumeration `esym` for `sym` over a domain `usym`. Evaluate the following,

```
esym?`msft`ibm
```

IBM changes its ticker symbol to "ibmx". Demonstrate the change you should make to your enumeration to reflect this.

Now, Kx Inc. goes public and is listed on the exchange. What must be done to make the following expression execute successfully,

```
esym,:`kx
```

Define,

```
usym:distinct sym
esym:`usym$sym
```

We find,

```
esym?`msft`ibm
1 0
usym[0]:`ibmx
esym
`usym$`ibmx`msft`ge`msft`ge`ibmx

`usym ?`kx
esym,:`kx
esym
`usym$`ibmx`msft`ge`msft`ge`ibmx`kx
```

C.6 Chapter 6

C.6.* Birthday Dictionary

Create a dictionary that maps the first names of three family members or friends, not including yourself, to their birthdays. Then add yourself to the dictionary. Create an equivalent column dictionary containing the same information, using column names ‘fname’ and ‘bday’. Retrieve the entry for your birthday from the column dictionary. Create the transpose of the column dictionary and retrieve your entry from it.

```
dbday:`omar`eybi`devon!1968.03.14 1992.04.22 1991.07.02

dbday[`zaphod]:1942.04.02

dbday[`zaphod]
1942.04.02

dccols:`fname`bday!(`omar`eybi`devon`zaphod;1968.03.14 1992.04.22 1991.07.02
1942.04.02)

dcols[;3]
`fname`bday!(`zaphod;1942.04.02)
```

C.6.* Char Dictionary

Create a dictionary that maps chars to their numeric representations and use it to translate a char to its int and vice ersa.

```
chars:(`char$v)!v:til 256
chars "*"
42
chars?100
"d"
```

C.6.* Dictionary Operations

Given the dictionary definitions.

```
d1:`a`b`c`d!10 20 30 40
d2:`d`c`e`f!400 300 500 600
```

we find,

```
d1,d2
`a`b`c`d`e`f!10 20 300 400 500 600
d2,d1
`d`c`e`f`a`b!40 30 500 600 10 20
1000+d1+d2
`a`b`c`d`e`f!1010 1020 1330 1440 1500 1600
10*d2+d1
`d`c`e`f`a`b!4400 3300 5000 6000 100 200
d1=d2
`a`b`c`d`e`f!000000b
(d1+d2)~d2+d1
0b
```

C.6.* Dictionary with Enumeration

Given the following dictionary definition, create an equivalent dictionary in which the range is replaced with an enumeration.

```
d:10 20 30 40 50 60 70 80!`a`c`b`a`b`b`c`a

u:distinct value d
(key d)!`u$value d
10 20 30 40 50 60 70 80!`u$`a`c`b`a`b`b`c`a
```

C.6.* Function Dictionary

Using only the multiplication operator, define three functions `p1`, `p2` and `p3` that return the first, second and third power (respectively) of a numeric argument. Create a dictionary that maps the symbols `` `first ``, `` `second `` and `` `third `` to the appropriate functions. Define a dyadic function `pow` whose first argument is numeric, whose second argument is a symbol and whose result is the number raised to the power described in the symbol.

```
p1:{x}
p2:{x*x}
p3:{x*x*x}

dpow:`first`second`third!(p1;p2;p3)

pow:{(dpow y) x}

pow[3;`second]
9
```

C.7 Chapter 7

C.7.* A Table Is a List

Given t defined as,

```
t:([] c:til 10)
```

we find,

```
t til 5
c
-
0
1
2
3
4
t where 10111b
```

```
c
-
0
2
3
4
select from t where 10111b
c
-
0
2
3
4
```

C.7.* Weight Watching - 1

You have decided to go on a strict diet. Part of the regimen is to monitor your weight daily. Define an empty table having columns dt containing a date and wt containing a float. Add some records with daily weigh-ins to the table using the , operator.

```
wts:([] dt:`date$(); wt:`float$())
wts,:(2007.01.01;191.5)
wts,:(2007.01.02;191.0)
wts,:(2007.01.02;192.0)
```

C.7.* Weight Watching - 2

Your personal diet is so successful that you decide to put your entire family on it. Define an enumeration over your family member's first names. Define a table to hold the daily weight observations with column fname containing an enum with the first name and the same remaining columns as in the previous exercise. Add some records with daily weigh-ins using the , operator.

```
fn:`Jeff`Hamlet`Othello`Orion
wts:([]fname:`fn$;dt:`date$(); wt:`float$())

wts,:(`fn$`Jeff;2007.01.01;191.5)
wts,:(`fn$`Hamlet;2007.01.01;12.2)
wts,:(`fn$`Othello;2007.01.01;15.7)
wts,:(`fn$`Orion;2007.01.01;7.3)
```

C.7.* Weight Watching - 3

Your family diet is so successful that your entire neighborhood wants to join your plan. Define a keyed table that has as primary key an integer id and has symbol columns with first and last name. Use the , operator to insert family members and neighbors into the keyed table using

sequential ids starting at 1001.

```
members:([id:0#0] lname:0#`; fname:0#`)

members,:(1001;`Borror;`Jeff)
members,:(1002;`Borror;`Hamlet)
members,:(1003;`Borror;`Othello)
members,:(1004;`Borror;`Orion)
members,:(1005;`Beeblebrox;`Zaphod)
members,:(1006;`Dent;`Arthur)
```

Define an observations table to have a foreign key `id` in the `members` table, and having the remaining date and weight columns as before. Add some records with daily weigh-ins using the `,` operator. Retrieve the last name, first name and weights for all observations.

```
wts:([]id:`members$;dt:`date$(); wt:`float$())

wts,:(`members$1001;2001.01.01;191.5)
wts,:(`members$1002;2001.01.01;11.5)
wts,:(`members$1003;2001.01.01;13.5)
wts,:(`members$1004;2001.01.01;7.2)
wts,:(`members$1005;2001.01.01;142.0)
wts,:(`members$1006;2001.01.01;167.0)

show select id.lname, id.fname, dt, wt from wts
lname fname dt wt
---------------------------------
Borror Jeff 2001.01.01 191.5
Borror Hamlet 2001.01.01 11.5
Borror Othello 2001.01.01 13.5
Borror Orion 2001.01.01 7.2
Beeblebrox Zaphod 2001.01.01 142
Dent Arthur 2001.01.01 167
```

C.7.* Pseudo-Join

Given a table `t` with symbol column whose values are in the domain of a lookup dictionary `dl` which maps the symbols to float values,

```
t:([] s:`a`c`a`b`a`c; v:10 20 30 40 50 60)
dl:`a`b`c!1.1 2.2 3.3
```

we find,

```
select s,v,dl[s] from t
s v s1
--------
```

```
a 10 1.1
c 20 3.3
a 30 1.1
b 40 2.2
a 50 1.1
c 60 3.3
```

C.8 Chapter 8

C.8.* Add a Constant Column

Given a table `t`, write an expression that adds a column `xxx` with the value 42 to `t`.

```
update xxx:42 from t
```

C.8.* Dynamic Foreign Key

Given a keyed table `o`,

```
o:([oid:1 2 3 4] pr:101 102 103 104)
```

and a table `e`,

```
e:([]onum:2 1 4; qty:10 20 30)
```

Write an expression that converts `e` to a table in which `onum` is a foreign key over `o`.

```
update `o$onum from e
```

Due to default column naming, this is the same as,

```
update onum:`o$onum from e
```

C.8.* Grouping

Given a table of the form,

```
t:([] sym:0#`; ti:0#0Nt; px:0#0f)
```

the following query retrieves prices grouped by hour for each symbol.

```
select px by sym, ti.hh from t
```

Write a query that retrieves prices grouped by 10 minutes for each symbol.

```
select px by sym, 600000 xbar ti from t
```

C.8.* First n Records By Column Value

Write a query that selects the first n records matching each value in a specific column. For example, given,

```
t:([] s:`i`m`i`i`m`g; v:1 2 3 4 5 1)
n:2
```

your result should include the values from rows 0, 1, 2, 4, 5 once each.

```
ungroup select (0;n) sublist v by s from t
s v
---
g 1
i 1
i 3
m 2
m 5
```

C.8.* Renaming a Single Column

Write an expression that renames any column in a table. For example, given,

```
t
c1 c2 c3
---------
a 10 1.1
b 20 2.2
c 30 3.3
```

we can rename column `c2` with,

```
show ?[`c2=c:cols t; `n2; c] xcol t
c1 n2 c3
---------
a 10 1.1
b 20 2.2
c 30 3.3
```

C.8.* Inverting a Keyed Table

Write a query that inverts a keyed table. Given,

```
kt:([k:`a`b`c`d`e] v:50 40 30 20 10)
```

we find,

```
exec v!k from kt
50| a
40| b
30| c
20| d
10| e
```

C.8.* Excluding Result Columns

For example, given,

```
t:([] a:1 2 3; b:`x`y`z; c:3.3 2.2 1.1)
```

The following query excludes columns `a` and `c`,

```
delete a,c from t
b
-
x
y
z
```

The functional form is,

```
![t;();0b;`a`c]
```

C.8.* Column Types

Write an expression that produces a column name to column type mapping for an arbitrary table or keyed table. For example, given,

```
t:([] c1:`a`b`c; c2:30 20 10)
```

we find,

```
exec c!t from meta t
c1| s
c2| i
```

C.8.* Resolving Foreign Keys

Given the table `t` and keyed table `kt`, write an expression that resolves the foreign key `f` to its underlying values.

```
kt:([k:`a`b`c] v:1 2 3)
t:([]f:`kt$`b`a`a`c; p:1 2 3 4)
```

```
update f:value f from t
```

Write a function that resolves all foreign keys in an arbitrary table or keyed table x.

```
{![x;();0b;c!value ,/: c:exec c from meta x where not null f]}
```

C.8.* Lab

One solution is given below.

1. Generate a list `S` of 1000 random 1-8 character stock symbols

```
S:`$`char$(`int$"a")+(1+1000?7)?\:26
```

2. Derive an expression that selects 5000 random symbols from `S`

```
5000?S
```

3. Derive an expression that generates 5000 random long integers between 1 and 1000000

```
1+5000?999999j
```

4. Create a table `q` that has a column `s` containing the result of (2) and a column `c` containing the result of (3).

```
q:([] s:5000?S; c:1+5000?999999j)
```

5. Derive an expression that extracts a list of the first letters from each item in the result of (2)

```
(string x)@'0
```

6. Create a keyed table `t` which maps a column `s` with the first letters of the quotes in `q` to a column `c` with the total number of quotes starting with that letter

```
t:select sum c by s:(string s)@'0 from q
```

7. Extract the count values from `t` into a vector `C`

```
C:exec c from t
```

8. For `n` equal to 6, derive an expression representing the size of `n` equal partitions of the original quote table

```
n:6
(sum C)%n
```

9. Create a list `B` representing the breaks for `n` equal buckets of the quotes

```
B:((sum C)%n)*1+til n-1
```

10. Derive an expression that compares the running sums of `C` to the partition size of (8)

```
(sums C)< (sum C)%n
```

11. Derive an expression that compares the running sums of `C` to all the bucket breaks in `B`

```
(sums C)</:B
```

12. Derive an expression that finds the indices `I` where the running sums of `C` cross the bucket breaks in `B`

```
I:sum flip (sums C)</:B
```

13. Derive an expression that show that letters where the partitions should occur to fill the buckets

```
(exec s from t) I
```

14. Create a function that takes the table `q` and the number of partitions `n` as parameters and returns the letters for the partition breaks

```
{[q;n]
// q is quote summary table
// n is number of partitions desired
// return letters for partition breaks
//
/ map first character to sum of its quote count
t:select sum c by s:(string s)@'0 from q;
/ extract count column for convenience
C:exec c from t;
/ create vector of n-1 equal bucket breaks
B:((sum C)%n)*1+til n-1;
/ indices where running sums cross breaks
I: sum flip (sums C)</:B
/ return letters for breaks
(exec s from t) I }
```

C.8.* Lab – Pivot Table

We begin with,

```
t:([]k:1 2 3 2 3; p:`a1`a2`a1`a3`a2; v:100 200 300 400 500)
```

```
t
k p v
--------
1 a1 100
2 a2 200
3 a1 300
2 a3 400
3 a2 500
```

and the desired result is,

```
pvt
k| a1 a2 a3
-| -----------
1| 100
2| 200 400
3| 300 500
```

Our approach will be to construct the solution for the specific table and then successively generalize it to handle a wider class of input tables and functionality.

1. Collect the distinct values of the pivot column `p` into a list `P`.

```
P: asc distinct t.p
/or
P:asc distinct t`p
/ or
P:asc exec distinct p from t
P
`a1`a2`a3
```

2. Write a query that extracts the value pairs for `p` and `v` grouped by `k`. (***Hint***: use exec)

```
exec p!v by k from t
1| (,`a1)!,100
2| `a2`a3!200 400
3| `a1`a2!300 500
```

3. Enhance the previous query to rectangularize the dictionaries by null filling missing values. This can be accomplished by evaluating the previous dictionaries on all values in `P`.

```
exec (p!v) P by k from t
1| 100
2| 200 400
3| 300 500
```

4. Enhance the previous query to produce record dictionaries in which the names of P are mapped to the corresponding values and the key column is appropriately named.

```
exec P!(p!v) P by k:k from t
k| a1 a2 a3
-| -----------
1| 100
2| 200 400
3| 300 500
```

5. Place the query in a function `pvt1` that takes the table and the column names as parameters and returns the pivot result.

```
pvt1:{[t;k;p;v]
// t is source table;k is key column; v is col that provides values
// p is pivot column
/ get unique values in pivot col
P:asc distinct t p;
/ construct pivot vectors as rectangular null-filled fields
/ exec dictionary mapping pivot entries to values
/ rectangularize by extracting all cols from dict
/ create named records
?[t;(); K!K:k,(); (!;`P;((!;p;v);`P))]}
```

6. Modify the function `pvt1` to obtain pvt2 that handles the case when the values of the pivot column are not symbols—e.g., integers—that would fail as column names. Test it on `tn`.

```
tn:([]k:1 2 3 2 3;p:(`a1;2;`a1;3;2);v:100 200 300 400 500)

pvt2:{[t;k;p;v]
// t is source table; k is list of key columns; p is pivot column
// v is col that provides values for result
/ replace pivot col values with their symbol equivalents
t:update p:`$string p from t;
/ get unique values in pivot col converted to symbol
P:asc distinct t p;
/ Ensure K is list
K:k,();
/ construct pivot vectors as rectangular null-filled fields
/ exec dictionary mapping pivot entries to values
/ rectangularize by extracting all cols from dict
/ create named records
?[t;(); K!K; (!;`P;((!;p;v);`P))]}

pvt2[tn;`k;`p;`v]
k| a1 2 3
-| -----------
1| 100
2| 200 400
3| 300 500
```

7. A problem with the function `pvt2` is that result columns whose names start with a non-alpha character are not accessible via a select (try it). Modify `pvt2` to obtain `pvt3` in which the columns resulting from pivot values starting with a non-alpha character

have "X" prepended to their name.

```
pvt3:{[t;k;p;v]
// t is source table;k is key column; v is col that provides value
// p is pivot column
/ get unique values in pivot col
/ replace pivot col values with their symbol equivalents
/ prepend "x" to those starting with non-aplha so they are accessible in select
t:update p:`${$[any (first x) within ("AZ";"az"); x; "X",x]} each string p from t;
/ get unique values in pivot col converted to symbol
P:asc distinct t p;
/ Ensure K is list
K:k,();
/ construct pivot vectors as rectangular null-filled fields
/ exec dictionary mapping pivot entries to values
/ rectangularize by extracting all cols from dict
/ create named records
?[t;(); K!K; (!;`P;((!;p;v);`P))]}

pvt3[tn;`k;`p;`v]
k| a1 X2 X3
-| -----------
1| 100
2| 200 400
3| 300 500
```

Next, we generalize to the case when the pivot table values are not unique to the key value. For example, for `tr` below, our current `pvts` misses the 1000 value.

```
tr:([]k:1 2 3 2 3 1; p:`a1`a2`a1`a3`a2`a1; v:100 200 300 400 500 1000)

pvts[tr;`k;`p;`v;]
k| a1 a2 a3
-| -----------
1| 100
2| 200 400
3| 300 500
```

We would like to apply an aggregate function such as `sum` to obtain,

```
k| a1 a2 a3
-| ------------
1| 1100
2| 200 400
3| 300 500
```

8. Write a functional query that creates a table `ta` from `t` by aggregating the `v` values by `p` using `sum`.

```
ta:?[t; (); (`p`k)!`p`k; (enlist `v)!enlist (sum;`v)]
```

9. Modify the function `pvt3` to obtain `pvtf` that takes an aggregate function `f` and applies it to `v` as part of the pivot process. Test it on `tr` using `sum`. How do we obtain the result of `pvts` from `pvtf`?

```
pvtf:{[t;k;p;v;f]
// t is source table; k is list of key columns; p is pivot column
```

```
// v is col that provides values for result
// f is aggregate function to be applied
// when there pivot col entries are not unique by key value
/ replace pivot col values with their symbol equivalents
/ prepend "x" to those starting with non-aplha so they are accessible in select
t:update p:`${$[any (first x) within ("AZ";"az"); x; "X",x]} each string p from t;
/ get unique values in pivot col converted to symbol
P:asc distinct t p;
/ Ensure K is list
K:k,();
/ use f to aggregate v grouped by pivot col, key col
t:?[t; (); (p,K)!p,K; (enlist v)!enlist (f;v)];
/ construct pivot vectors as rectangular null-filled fields
/ exec dictionary mapping pivot entries to values
/ rectangularize by extracting all cols from dict
/ create named records
?[t;(); K!K; (!;`P;((!;p;v);`P))]}

pvtf[tr;`k;`p;`v;sum]
k| a1 a2 a3
-| ------------
1| 1100
2| 200 400
3| 300 500

pvtf[tr;`k;`p;`v;first]
k| a1 a2 a3
-| -----------
1| 100
2| 200 400
3| 300 500
```

10. Check that the function `pvtf` accepts a list of symbols for the key column and creates a compound key. Test it against `tK`.

```
tK:([]k1:1 2 3 2 3 1 1;k2:10 20 30 40 50 60 10; p:`a1`a2`a1`a3`a2`a1`a1;
v:100 200 300 400 500 1000 10000)

pvtf[tK;`k1`k2;`p;`v;sum]
k1 k2| a1 a2 a3
-----| -------------
1 10| 10100
1 60| 1000
2 20| 200
2 40| 400
3 30| 300
3 50| 500
```

11. Modify the function `pvtf` to obtain `pvtfw` that accepts a string argument represening a valid where clause for the table and uses it to constrain the pivot. Test it against `tK` with the clause "k1<>2".

```
pvtfw:{[t;k;p;v;f;w]
// t is source table; k is list of key columns; p is pivot column
// v is col that provides values for result
// f is aggregate function to be applied
// when there pivot col entries are not unique by key value
// w is string containing valid where clause for t
/ constrain t by w
t:?[t; $[""~w;(); enlist parse w]; 0b; ()];
/ replace pivot col values with their symbol equivalents
/ prepend "x" to those starting with non-aplha so they are accessible in select
t:update p:`${$[any (first x) within ("AZ";"az"); x; "X",x]} each string p from t;
/ get unique values in pivot col converted to symbol
P:asc distinct t p;
```

```
/ Ensure K is list
K:k,();
/ use f to aggregate v grouped by pivot col, key col
t:?[t; (); (p,K)!p,K;(enlist v)!enlist (f;v)];
/ construct pivot vectors as rectangular null-filled fields
/ exec dictionary mapping pivot entries to values
/ rectangularize by extracting all cols from dict
/ create named records
?[t;(); K!K; (!;`P;((!;p;v);`P))]}}


pvtfw[tK;`k1`k2;`p;`v;sum;"k1<>2"]
k1 k2| a1 a2 a3
-----| ------------
1 10| 10100
1 60| 1000
3 30| 300
3 50| 500
```

12. Modify `pvtfw` to allow the user to pass an empty list as the aggregate or the where clause. In the former case, use `first` as the default aggregate; in the latter, perform no constraint. Test against `tK` with empty list arguments for `f` and `w`.

```
pvtfw:{[t;k;p;v;f;w]
// t is source table; k is list of key columns; p is pivot column
// v is col that provides values for result
// f is aggregate function to be applied
// when pivot col entries are not unique by key value; defaults to first
// w is string containing valid where clause for t
// allow empty list or empty char list
/ constrain t by w
t:$[""~"c"$w; t; ?[t;enlist parse w;0b;()]];
/ replace pivot col values with their symbol equivalents
/ prepend "x" to those starting with non-aplha so they are accessible in select
t:update p:`${$[any (first x) within ("AZ";"az"); x; "X",x]}each string p from t;
/ get unique values in pivot col converted to symbol
P:asc distinct t p;
/ make first the default aggregate
f:$[()~f;first;f];
/ Ensure K is list
K:k,();
/ use f to aggregate v grouped by pivot col, key col
t:?[t; (); (p,K)!p,K;(enlist v)!enlist (f;v)];
/ construct pivot vectors as rectangular null-filled fields
/ exec dictionary mapping pivot entries to values
/ rectangularize by extracting all cols from dict
/ create named records
?[t;(); K!K; (!;`P;((!;p;v);`P))]}


pvtfw[tK;`k1`k2;`p;`v;();()]
k1 k2| a1 a2 a3
-----| ------------
1 10| 100
1 60| 1000
2 20| 200
2 40| 400
3 30| 300
3 50| 50
```

13. Define `pvt` as a projection of `pvtfw` by defaulting the lst two arguments.

```
pvt:pvtfw[;;;;();()]
```

C.9 Chapter 9

C.9.* Depth of a List

Write a function that returns the depth of an arbitrary list and does not use `do` or `while`.

Recall that the sign of the type of an entity indicates whether it is an atom or a list. Use `each` to recurse over the items of the list.

```
depth:{$[0>type x; 0; 1+max depth each x]}
```

C.10 Chapter 10

C.10.* Homicidal Partner

Assume that h is an open handle to another q process. Write an expression that kills the remote process.

```
h "value \"\\\\\\\\\\""
```

C.11 Chapter 11

C.11.* Cleaning House

Write an expression that truncates (i.e., removes all rows of) all tables in the default context. Generalize to any context.

```
@[`.;tables `.;0#]
```

```
{@[x;tables x;0#]}
```

C.11.* All Tables

Write an expression that produces a dictionary whose domain is all the non-reserved contexts and whose range is the list of tables in each of the contexts.

```
(`$c)!value each "\\a ",/: c:".",'c where 1<>count each c:string each `,key `
```