

2011 International APL Programming Contest

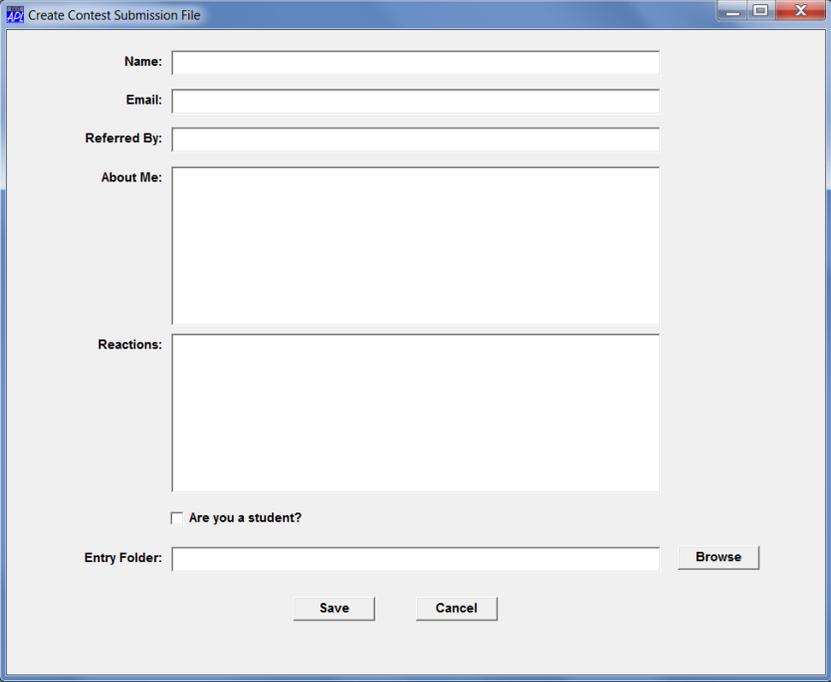
Sponsored by: Fiserv (USA), SimCorp (Denmark), APL Italiana (Italy), and Dyalog (UK)

Welcome!

Thank you for your interest in participating in this year's contest! This is the third International APL Programming contest, and the format has been updated from previous years. This year's contest consists of 5 problems across a diverse set of disciplines including airline route analysis, DNA sequencing, image processing, and text searching. Each problem has 2 or more tasks. We would like to thank last year's winner, Ryan Tarpine, for his help in developing this year's problems.

How To Submit Your Entry

1. Download the Contest2011.dws workspace from the contest webpage.
2. There are 5 namespaces named **Problem1** through **Problem5** each of which contains the required data for its problem as well as stub functions that you can use to start coding your solutions. You can use the supplied stub functions, or code your own as long as you use the names as described in the task descriptions.
3. Make sure you save the workspace to save your work.
4. When you're satisfied with your solutions, run the function **#.SubmitMe**. Enter the requested information and click the Save button. This will create an APL script file named `Contest2011.dyalog` in the directory you specified.



The screenshot shows a dialog box titled "Create Contest Submission File". It contains the following fields and controls:

- Name: [Text Input]
- Email: [Text Input]
- Referred By: [Text Input]
- About Me: [Text Area]
- Reactions: [Text Area]
- Are you a student?
- Entry Folder: [Text Input] [Browse]
- [Save] [Cancel]

5. Email the `Contest2011.dyalog` file to Contest2011@dyalog.com.
6. Questions about the contest can be sent to Contest2011@dyalog.com

Good luck!

Problem 1 – Take To The Air

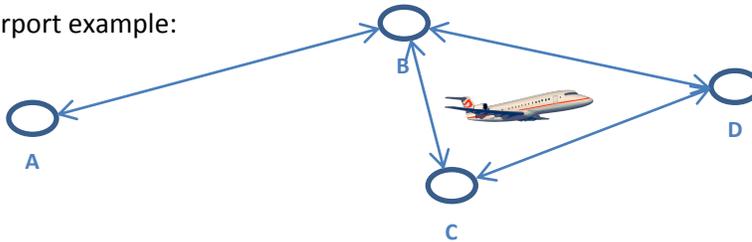
Background

You've been put in charge of both the marketing and financial planning of a small airline. You have been supplied a 2 column table of the current segments (pairs of airports) between which your airline provides service. The order of the columns is not significant as your airline provides service in both directions of each segment. This table can be found in the contest workspace in the variable `#.Problem1.Segments`. A table of all of the airports is provided in the variable `#.Problem1.Airports`, whose first column contains the airport name and the second and third columns contain the latitude and longitude for the airport.

Task 1 – Build an adjacency matrix for the segments.

The first step is to put the data into a format that's easier to work with. Write a function named `adjacency` which takes the Segments and Airports table as its arguments and returns an $M \times M$ Boolean matrix representing the connectivity of the segments where M is the number of distinct airports.

In this 4 airport example:



TestSegments	TestAirports
A B	A 33 56 33 N 118 24 29 W
B C	B 41 58 46 N 87 54 16 W
B D	C 40 38 23 N 73 46 44 W
C D	D 51 28 39 N 0 27 41 W

TestAirports	adjacency	TestSegments	
0	1	0	0
1	0	1	1
0	1	0	1
0	1	1	0

The adjacency matrix should have a 1 in row i and column j if there is a segment between the i^{th} and j^{th} airports in the airports table.

Task 2 – Getting There From Here

From a marketing viewpoint, you want to advertise every possible destination that a passenger can reach, regardless of whether he/she must take one, two, three, or even more flights. For example, if you have a flight from airport A to airport B, and another flight from airport B to airport C, then you want to advertise that a passenger can fly to airport C from airport A. Write a function `allTrips` which takes the adjacency matrix from task 1 and returns another $M \times M$ Boolean matrix which has a 1 in the i^{th} row and j^{th} column if and only if someone can travel between the i^{th} airport and the j^{th} airport using one or more flights. In our example from above, you can get to any airport from any other airport.

```

allTrips TestAirports adjacency TestSegments
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

```

Task 3 – Minimize cost

Due to rising fuel costs and budget constraints, you would like minimize cost by reducing the number of segments your airline services while still maintaining all the destinations a passenger can currently reach. In the example above, any of one the segments B-C, B-D, or C-D could be removed without affecting the airports one can travel to, though they may have to travel through more segments to get to their destination, but which one will minimize the cost? This task has 2 sub-tasks.

Subtask 3a – Go The Distance

The latitude and longitude values in the Airports table can be used to calculate each segment’s length. The formula for distance to be used is:

$$distance = 6371 \times \cos^{-1}((\cos(Long1 - Long2) \cos(Lat1) \cos(Lat2)) + \sin(Lat1) \sin(Lat2))$$

where Lat1, Lat2 and Long1,Long2 are the latitudes and longitudes for the two points (6371 is the radius of the Earth in kilometers). The latitudes and longitudes for each airport are found in the last 2 columns of the Airports table. You will need to convert the lat/long from its degree/minute/second format into decimal degree. Latitudes in the Southern hemisphere and longitudes in the Western hemisphere should have negative values. The trigonometric function in APL (∘) uses values expressed in radians. Your task is to write a function **distance** which takes a lat/long for its left and right arguments and returns their distance apart in kilometers. You may want to write a function to convert from degree/minute/second format to decimal degree. For example:

```

dms2dd 34 56 42 'S' A dms to decimal degrees
-34.944999999999999
((34 56 42 'S')(138 31 50 'E')) distance ((3 32 35 'N')(76 22 53 'W'))
15004.00859491403

```

Subtask 3b – Reduce Segments

In simple terms, the cost of a segment is proportional to its distance. Your task is to minimize the overall cost. To arrive at a minimum cost Armed with your adjacency matrix and Your task is to write a function, **reduceSegments** which takes the Airports and Segments tables as its arguments and returns two Segments tables – the original Segments table with a distance column added and the new, reduced, Segments table. Using the example tables:

```

TestAirports reduceSegments TestSegments
A B 15004.00859491403 A B 15004.00859491403
B C 8784.45815117948 B C 8784.45815117948
B D 14987.1840396508 C D 8812.504030110875
C D 8812.504030110875

```

Problem 2 – What’s in a Name?

Background

You have been hired to monitor blog and forum references on the internet to your employer’s company name and executives’ names. You know a simple internet search will not find what you're looking for because people often misspell the names. You decide to search the text yourself looking for mentions of the names allowing for a certain number of mismatches.

Task 1 – Simulate Misspellings

You want to begin by simulating examples that you will test your algorithm on. You plan to take company documents, introduce misspellings, and see whether you can still find the names. Write a function **addNoise** which takes two arguments. The right argument is a string to which the misspellings will be added. The left argument is a number between 0 and 1 representing the rate. **addNoise** should, with the probability specified by the left argument, replace each letter in the string with the letter 'X'. For example, if the rate is 0.05, then on average 5% of the letters should be replaced with the letter X.

Example:

```
0.1 addNoise 'Jack Johnson'  
Xack JohXson
```

Task 2 – Matching Mismatches

Write a function **patFind** which takes two arguments. The left argument is text in which to look for the pattern. The right argument is a vector of two elements, a pattern and a tolerance. The pattern is the word or phrase to search for in the larger text. The search should be case insensitive. The tolerance is the maximum number of mismatches to allow. Your function should return a table whose first column contains the positions in the text where the pattern matches and second column contains the matched pattern from the text.

Example:

```
'I think Jakc Jonnson is the greatest' patFind 'Jack Johnson' 3  
9 Jakc Jonnson  
'I think Jakc Jonnson is the greatest' patFind 'Jack Johnson' 2
```

The contest workspace contains a character vector **#.Problem2.SampleText** which contains a number of spelling variations of the name "VanDerHeusen". Task 2 will be judged on the results of searching the sample text for (mis)matches on "VanDerHeusen" as follows:

```
SampleText patFind 'VanDerHeusen' 3
```

Problem 3 – It's All About Image

Background

You've recently formed a startup with some associates to develop and sell an image manipulation program in APL. You have an important upcoming meeting with potential investors and want to develop a simple demo to convince them how using APL gives you an edge in the market. You decide to present and demonstrate code to scale and blur images, confident that when they see how appropriate APL is, they will invest heavily.

The color of a pixel is typically represented by three numbers, representing the amount of red, green, and blue in the pixel. Each number ranges between 0 and 255, where 0 means none of that color while 255 means full intensity. In this manner, the color black is represented by the three numbers (0,0,0), red is (255,0,0), green is (0,255,0), and white is (255,255,255). One way of computing a single number between 0 and 255 which represents a shade of gray similar to the original color is to simply find the average of the three numbers.

These three numbers are often stored as one number by combining them using the formula $256^2 * RED + 256 * GREEN + BLUE$. So black becomes 0, red becomes 16711680, green becomes 65280, and white becomes 16777215. In this manner, all colors can be represented by a number between 0 and 16777215.

The contest workspace contains a sample picture in the variable `#.Problem3.Logos`, and a simple function `#.Problem3.show`, that displays the picture formed by a matrix of pixels in RGB format passed as the argument. You can use `Logos` to see the effects of your results from the tasks below.

```
show Logos A displays Logos
```



Task 1 – Shades of Gray

For simplicity, you decide to work first with grayscale only (not the full range of colors). So you begin by writing a function to convert any color to a simple grayscale value between 0 and 1, where 0 means black, 1 means white, and numbers in between represent various shades of gray.

Write a function `toGray` which takes an array of these numbers representing all three colors, splits it into its three parts, takes the average, and then divides this single number by 255 to yield a number between 0 and 1.0 representing the gray level of that color.

Example:

```
toGray 0 16711680 65280 16777215
0 0.3333333333 0.3333333333 1
```

Task 2 – Back to RGB

After converting an image to grayscale with `toGray`, we need to convert the values between 0 and 1 back into RGB values so that they can be displayed by `show`. Write a function `toRGB` which, given an array of numbers between 0 and 1, converts them to integers (N) between 0 and 255 and returns an array of numbers between 0 and 16777215 representing a pixel whose red, green, and blue are all equal to the values of N.

Example:

```
toRGB 0 (÷3) 1
0 5592405 16777215
toGray toRGB ÷3
0.3333333333
toRGB toGray 5592405
5592405
show toRGB toGray Logos
```



Task 3 – Off The Scale

To scale an image, write a function `scale` which takes two arguments. Its left argument should be an array of 2 integers - the first element is the vertical scaling factor (i.e. the number of copies of each row to make) and the second element is the horizontal scaling factor (i.e. number of copies of each column to make). The right argument of `scale` is the image to scale.

Example:

```
2 3 scale 3 3p1+i9
1 1 1 2 2 2 3 3 3
1 1 1 2 2 2 3 3 3
4 4 4 5 5 5 6 6 6
4 4 4 5 5 5 6 6 6
7 7 7 8 8 8 9 9 9
7 7 7 8 8 8 9 9 9

show 1 4 scale Logos
```



When the sum of the values in a convolution matrix is greater than 1, it has a brightening effect on the colors of the image. So with a matrix where the values add up to 8, the resulting image is going to be roughly 8 times brighter than the original image. To compensate for this, your code should normalize the convolution matrix by dividing it by the sum of its values or 1 if the sum is less than or equal to 0.

Subtask 4a – Gray Convolution

For image processing, values that fall outside the valid range need to be adjusted. Values produced by **toGray**, have a range of 0 to 1. So, any value in the result which is less than 0 should be set to 0. Similarly, any value greater than 1 should be set to 1. Write a function **convolve** which takes two arguments where the left argument is an M x N convolution matrix (where M and N are odd positive integers) and the right argument is a two-dimensional matrix of grayscale values between 0 and 1. **convolve** will return a new matrix of grayscale values between 0 and 1 which are the result of the convolution.

Example:

```

C ← P ← toGray 5 5 ↓ Logos A grab an "interesting" part of Logos
1           1           1           1           1
0.4522875817 0.4993464052 0.4784313725 0.4575163399 0.5045751634
0.5267973856 0.9385620915 0.968627451  0.9529411765 0.9869281046
0.4522875817 0.9869281046 0.9960784314 0.9934640523 0.9934640523
0.4718954248 0.9843137255 0.9633986928 0.9947712418 0.9895424837

```

```

C convolve P A using C, the "emboss" convolution matrix from above
0           0.5281045752 0           0           0
0           0.4013071895 0.3111111111 0.4104575163 0
1           1           1           1           1
0.385620915 1           1           1           0
1           1           1           1           0.4862745098

```

Show toRGB C convolve Logos A embossed gray-scale logos



Subtask 4b – Color My World

Your next task is, you guessed it, to apply convolution to full color images. The technique is similar to Subtask 4a, except that you will apply the convolution to each of the red, green, and blue components of a pixel. Write a function, **convolveRGB** which takes a convolution matrix as its left argument and a matrix of RGB values and returns a matrix of new RGB values which are the result of the convolution.

Example:

```
⌘←C←3 3ρ0 1 0 1 -4 1 A "edge detect" convolution matrix
0 1 0
1 -4 1
0 1 0
```

Show C convolveRGB Logos



Task 5 – Take a Wild Gauss

A popular type of blur is called the Gaussian blur. It is performed by using a convolution matrix generated by a Gaussian function. Gaussian matrices depend on a value, σ , that determines how strong the blur is. In theory, each pixel in the resulting image depends on every pixel in the original image, but in practice the effect from distant neighbors is too small to be important, so you decide to make your Gaussian matrices of size: $N = 2 \times \lceil 3\sigma \rceil + 1$ where $\lceil 3\sigma \rceil$ is the ceiling of 3σ .

The formula for each cell of the Gaussian matrix is:

$$G_{i,j} = \frac{1}{2\pi\sigma^2} e^{-\frac{(i-\frac{N-1}{2}-1)^2+(j-\frac{N-1}{2}-1)^2}{2\sigma^2}}$$

The value in the numerator of the power of e is simply the square of the distance from the center of the matrix to the cell i,j .

Example:

```
5 ⌘ gaussianMat 1
0.00002 0.00024 0.00107 0.00177 0.00107 0.00024 0.00002
0.00024 0.00292 0.01306 0.02154 0.01306 0.00292 0.00024
0.00107 0.01306 0.05855 0.09653 0.05855 0.01306 0.00107
0.00177 0.02154 0.09653 0.15915 0.09653 0.02154 0.00177
0.00107 0.01306 0.05855 0.09653 0.05855 0.01306 0.00107
0.00024 0.00292 0.01306 0.02154 0.01306 0.00292 0.00024
0.00002 0.00024 0.00107 0.00177 0.00107 0.00024 0.00002
```

show (gaussianMat 1) convolveRGB Logos A Gaussian blur



Time For a Little Fun...

This section has nothing to do with the contest, but since we're dealing with graphics, we thought it might be interesting to show a few more examples.

First some simple APL array manipulations...

```
show ⍉Logos ⍲ transpose
```



```
show ⍉Logos ⍲ vertical axis
```



```
show ⍉Logos ⍲ horizontal axis
```



```
show (-1 1 ⍲Logos)⍉Logos ⍲ skew
```



```
show 256⍲256 256 256⍲Logos ⍲ RGB swap
```



Now for a few more convolutions...

Sharpen

```
show (3 3⍲0 -1 0 -1 5 -1) convolveRGB Logos
```



Blur to the 50th power

```
show (3 3⍲1) (convolveRGB*50) Logos
```



Use the power operator to apply a Gaussian blur twice

```
show (gaussianMat 1) (convolveRGB*2) Logos
```



Gaussian blur to the 50th power

```
show (gaussianMat 1) (convolveRGB*50) Logos
```



We now return you to your regularly scheduled contest...

Problem 4 –I Must Have Gained Weight Because My Genes are Binding

Background

You have been hired by a team of biologists to help them look for places where transcription factors bind in a DNA sequence. Transcription factors are proteins which recognize specific sequences of DNA and bind there in order to regulate which genes in a cell are turned on or off. DNA, in reality a sequence of molecules called nucleotides, is modeled computationally as a string of As, Cs, Gs, and Ts. Each factor binds to a family of similar sequences. For example, the factor called Su(H) is known to bind to the following sequences¹:

```
TCGTGAGAAACGAGAC
ATGTGAGAAACCGAGT
GCGTGGGAACCGAGCT
TCGTGGGAACCCCGGA
GTGTGAGAAACTTACT
CTGTGGGAACCTGGTAG
GTGTGGGAAACTACAA
CTGTGGGAACATTGCT
CTGTGGGAACGGAAAG
TCGTGGGAAACACTTT
```

They are all similar, but none of these examples are exactly alike. In the examples below, the variable **#.Problem4.Train** will refer to this 10 x 16 matrix.

Task 1 – Freq Out!

A standard representation of the sequences a transcription factor is known to bind to is a frequency matrix. A frequency matrix has 4 rows, one for each type of nucleotide (A, C, G, T), and one column for each character of the pattern to recognize. In the example above, the frequency matrix has 16 columns. The number in the i^{th} row and the j^{th} column is equal to the number of examples for which the j^{th} column was equal to the i^{th} nucleotide.

The frequency matrix for the example sequences above is:

```
1 0 0 0 0 3 0 10 10 5 1 1 4 3 4 2
3 4 0 0 0 0 0 0 0 5 7 2 2 1 3 1
3 0 10 0 10 7 10 0 0 0 1 4 2 4 2 2
3 6 0 10 0 0 0 0 0 0 1 3 2 2 1 5
```

Notice that the sum of the numbers in each column is 10, the number of examples.

Write a function **freqMat** which, given a M x N matrix of As, Cs, Ts, and Gs, returns a 4 x N frequency matrix.

Task 2 – A Weighty Position

To use a frequency matrix to look for sequences similar (but not necessarily identical) to the examples, it is first converted to a *position weight matrix* (PWM). Write a function **makePWM** which takes a 4 x N frequency matrix F and returns a 4 x N position weight matrix M generated by the following formula:

¹ The binding sequence data was taken from the JASPAR database: Bryne JC, Valen E, Tang MH, Marstrand T, Winther O, da Piedade I, Krogh A, Lenhard B, Sandelin A. JASPAR, the open access database of transcription factor-binding profiles: new content and tools in the 2008 update.

$$M_{ij} = \log \frac{F_{ij} + 1}{\sum_{k=1}^4 (F_{kj} + 1)}$$

This essentially adds 1 to each element of the frequency matrix, then normalizes each column so each column sums to 1, and finally takes the logarithm of each.

Example:

```
1¶makePWM freqMat Train
-1.9 -2.6 -2.6 -2.6 -2.6 -1.3 -2.6 -0.2 -0.2 -0.8 -1.9 -1.9 -1.0 -1.3 -1.0 -1.5
-1.3 -1.0 -2.6 -2.6 -2.6 -2.6 -2.6 -2.6 -2.6 -0.8 -0.6 -1.5 -1.5 -1.9 -1.3 -1.9
-1.3 -2.6 -0.2 -2.6 -0.2 -0.6 -0.2 -2.6 -2.6 -2.6 -1.9 -1.0 -1.5 -1.0 -1.5 -1.5
-1.3 -0.7 -2.6 -0.2 -2.6 -2.6 -2.6 -2.6 -2.6 -2.6 -1.9 -1.3 -1.5 -1.5 -1.9 -0.8
```

These values were rounded to one decimal place for display only; your function should not round its results.

Task 3 – Scoring Sequences

To score a sequence using a PWM, you sum the values for each letter according to the columns they are in. The formula for a PWM M and a sequence S is:

$$score = \sum_{j=1}^N M_{S,j}$$

Write a function **score** which takes a left argument of a 4 x N PWM and a right argument of a string of As, Cs, Ts, and Gs. If the string is of length L, it should return a vector of length L - N + 1. The i^{th} element of this vector must be the score of the substring of length N starting at position i in the string.

Example:

```
1¶(makePWM freqMat train) score 'CCATCGTGGGAAACACTTTTCGAA'
-28.8 -27.2 -24.3 -13.5 -24.4 -20.4 -26.8 -27.2
```

Again, these values were rounded to one decimal place for display only; your function should not round its results.

Task 4 – Top It Off

Write a function **top5** which takes two arguments like **score**, a 4 x N PWM and a string. It should return a matrix of size 5 x N which contains the 5 highest scoring subsequences of the string, in order from highest to lowest score. You may assume it will only be passed strings of length N+4 or higher

Example:

```
(makePWM freqMat train) top5 'CCATCGTGGGAAACACTTTTCGAA'
TCGTGGGAAACACTTT
GTGGGAAACACTTTTCG
ATCGTGGGAAACACTT
CGTGGGAAACACTTTC
TGGGAAACACTTTTCGA
```

Problem 5 – Go Find It Yourself

Background

To accelerate your learning of APL, you decide to make your own custom search engine for articles published in *Vector*, the journal of the British APL Association. For simplicity's sake, you decide to start off by only searching article titles. A vector of titles is found in the variable `#.Problem5.Titles`.

Task 1 – A Token Effort

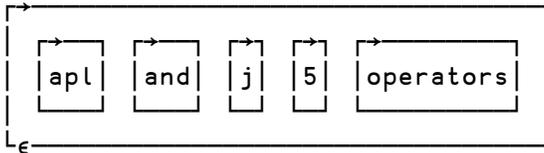
Write a function `tokenize` which takes as its only argument a string and returns a list of the tokens present in that string, converted to lowercase. A token is a sequence of letters and numbers. Punctuation is ignored.

Examples:

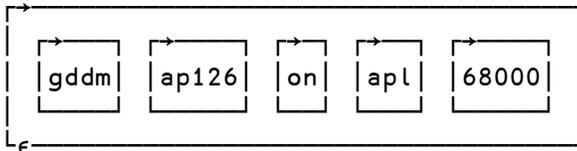
```
+t←tokenize 'APL and J (5): Operators - Inner Product, Each, Commute, Compose'
apl and j 5 operators inner product each commute compose
pt
```

10

```
]display2 5↑t
```



```
]display tokenize 'GDDM / AP126 on APL.68000'
```



Task 2 – It's in the Dictionary

Write a function `makeDict` which takes as its only argument a list of titles (a vector of character vectors) and returns a list of the distinct tokens present in any of the titles. We'll call this list a dictionary.

Example:

```
3 5p 15↑ makeDict 'Appropriate Use of APL in AI (1988)' 'Why GKS is Unsuitable
...' 'APL Experiences and Visual Basic for Windows'
appropriate use of apl in
ai 1988 why gks is
unsuitable experiences and visual basic
```

Note that "APL" is present in two of the titles but is present only once in the result. The order of the words in the result is not important.

² `]display` is a user command which displays the structure of the result of an APL expression

Task 3 – Count 'Em Up

Write a function **dictCount** which takes a dictionary left argument and a string right argument. The result should be a vector of the same length as the dictionary, but the value of each element is the number of times the corresponding word in the dictionary appears in the given string.

Example:

```
( 'and' 'apl' 'be' 'experiences' 'not' 'or' 'the' 'to' ) dictCount 'A to Be or
not A to Be?'
0 0 2 0 1 1 0 2
```

The result indicates that 'be' appears twice, 'not' appears once, 'or' appears once, and 'to' appears twice. None of the other words in the dictionary are found in the string, so the values for their positions are 0. Words in the string that are not found in the dictionary are not involved in the result.

Task 4 – What's Your Cosine?

One way of judging the similarity of two vectors of this model is called the *cosine similarity*. This measure treats each list as a vector in a multidimensional space and finds the cosine of the angle between the vectors. If the vectors are equal, the angle is 0 so the cosine of the angle is 1. If the vectors are orthogonal, sharing no words in common, then the angle is 90 degrees, yielding a cosine of 0.

The cosine of the angle is found by taking the dot product of the two vectors and then dividing by the magnitude of each. The magnitude of a vector is the square root of the sum of the square of each of its elements. That is:

$$similarity = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Write a function **cosine** which takes two equal-length vectors as its two arguments and returns the cosine similarity between the two vectors.

Examples:

```
0 1 cosine 0 1
1
1 1 cosine 0 1
0.7071067812
1 0 cosine 0 1
0
1 2 3 cosine 4 5 6
0.9746318462
```

Task 5 – Seek and Ye Shall Find

You're now going to use **cosine** to compute the relevance of each article title to a query in order to implement your search engine. Write a function **seek** which takes a left argument which is the list of titles to search and the right argument is a query string. **seek** should use the titles to create a dictionary, and then use this dictionary to convert all of the titles and the query into vectors. It should then return up to the top 10 titles in terms of their

cosine similarity to the query. It should only return titles whose similarity score is above 0. If none of the words in the query are found in the dictionary, **seek** should return an empty result.

Examples:

```
⋄Titles search 'suduko'  A misspelled query results in no hits
```

```
⋄Titles search 'sudoku'
```

```
Sudoku with Dyalog APL
```

```
A Sudoku Solver in J
```

```
⋄Titles search 'Error Trapping'
```

```
Error Trapping Tutorial for APL.68000
```

```
Error Trapping Tutorial for APL*PLUS
```

```
Error Trapping Tutorial for IPSA APL
```

```
Error Trapping Tutorial in Dyalog APL
```

```
Tutorial on Error Trapping in APL2/PC
```

```
Letter: Chaos - Computer Error not to Blame
```

Revision Summary

16 May 2011

The original version of the problem descriptions was found to have two errors in the description of Problem 3.

- The example output **convolve** for Subtask 4a was incorrect and has been corrected.
- The Gaussian matrix formula in Task 5 had a misprint and has been corrected.