

2012 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 fourth International APL Programming contest and consists of 4 problems. Each problem has 2 or more tasks. We've tried to select problems that are challenging, diverse, and hopefully fun as well.

Problem 1 has 5 tasks and explores the world of Texas Hold'em Poker first by using APL to evaluate poker hands and then calculate winning percentages.

Problem 2 has 3 tasks involving Roman numerals and includes building a user defined operator.

Problem 3 adds an APL twist to the classic problem of manipulating the digits 1-9 to produce a desired result.

Problem 4 involves solving the Japanese puzzle "Nurikabe". Nurikabe is one of the scores of games and logic puzzles from Japanese publisher Nikoli Co. Ltd.

How To Submit Your Entry

You may use any APL system to develop your solution. Whatever APL system you use, you must supply your solution in the form of an APL script file named **Contest2012.dyalog** which must be able to be loaded and executed from Dyalog APL.

- If you choose to use Dyalog APL for Windows for the contest...
 1. You may download the **Contest2012.dws** workspace from the contest webpage. The workspace contains 5 namespaces named **Problem1** through **Problem5** each of which contains any 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 and syntax as described in the task descriptions.
 2. Make sure you save the workspace to save your work. Use the **)save** system command to save your workspace.
 3. 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 **Contest2012.dyalog** in the directory you specify.

-
-

Screen Shot of the GUI for #.SubmitMe

- If you chose to develop your solutions using an APL system other than Dyalog APL...
 1. You must still submit your solutions in an APL script file named **Contest2012.dyalog**
 2. Dyalog has made available a template **Contest2012.dyalog** file for you to download from the contest webpage.
 - i. Enter the pertinent information, your name, email, etc. in the appropriate places in the script file.
 - ii. You may modify the stub functions and/or provide your own code as necessary to implement your solutions provided that your solutions conform to the name and syntax descriptions found for each problem. Additional functions that you supply as a part of your solution may be named however you choose.
 - iii. Do not change the namespace structure of the file.
- The **Contest2011.dws** workspace and **Contest2011.dyalog** template file contain stub functions for each of the tasks described in the contest. The stub functions are written as traditional APL functions. If you prefer to implement your solution using dfns (dynamic functions), you are welcome to do so!
- Email the **Contest2012.dyalog** file to Contest2012@dyalog.com

Other Information

- Questions about the contest can be sent to Contest2012@dyalog.com
- If necessary, updated information regarding the contest, e.g. problem clarifications or corrections, will be posted on the contest website. Therefore it is recommended that you check the website periodically.

Good luck!

Problem 1 – Hold'em Up!

Background

Over the last 10 or so years, the movie "Rounders", the World Poker Tour, and the World Series of Poker have contributed to the dramatic increase in the popularity of the poker game known as Texas Hold'em. Either that, or the fact that top prizes in poker tournaments can reach over a million dollars/euros 😊. Problem 1 has 5 tasks related to Texas Hold'em. If you are not familiar with Texas Hold'em specifically, or poker in general, we've included some basic information following the task descriptions.

Task 1 – A Value Judgment

Your first task is to write a function named **HandValue** which will take a character vector right argument which represents a poker hand and will return an integer representing the value of the hand. Better hands should have higher values. The right argument should be in the form 'RS RS RS RS RS' where R is the rank of the card and S is the suit as follows:

- Suits are any of 'SHDC' for Spades, Hearts, Diamonds, and Clubs respectively.
- Ranks are any of 'AKQJT' for Ace, King, Queen, Jack, Ten (10), and '98765432'
- Hands should be case insensitive such that 'QD', 'qD', 'qd', and 'Qd' all represent the Queen of Diamonds.
- Spaces separating the cards are allowed so that 'QCQD9H7D2C' and 'QC QD 9H 7D 2C' are considered to be equivalent.

Sample syntax: `value ← HandValue 'QC QD 9H 7D 2C'`

How you calculate the value for a hand is up to you, but the judges will look more favorably on efficient, array oriented solutions.

Task 2 – Nothing But The Best

The next task is to determine the best 5 card poker hand from the 7 cards represented by a player's 2 hole cards and the 5 community cards. Write a function, **BestHand** which takes as its left argument a character vector representing the community cards and a character vector representing a player's hole cards as its right argument and returns the best 5 card poker hand that can be made from the 7 cards.

Example:

```
best ← 'QS 8H 7S QH AC' BestHand 'AD 7C'  
best  
AD AC QS QH 8H
```

Task 3 – Showdown

Write a function named **Showdown** that takes as its left argument a character vector representing the community cards and as its right argument a vector of character vectors representing the players' hole cards. Showdown should return a 2 column matrix with as many rows as the number of hands in the right argument. Column 1 contains a character vector representing the best 5 card hand for each player. Column 2 contains a numerical

ranking for that hand relative to the other hands. Note – hands that tie in value should receive the same rank. Column 3 contains the number of the player's hole cards that were used to make the best poker hand.

Example:

```

      'QS 8H 7S QH AC' Showdown 'AD 7C' 'AS 4D' 'QD 8S' '3S 2H'
AD AC QS QH 8H    2    1
AS AC QS QH 8H    2    1
QD QS QH 8S 8H    1    2
QS QH AC 8H 7S    3    0

```

Task 4 – Am I Nuts?

The "nuts" are the player's hole cards that will make the best possible poker hand based on the community cards that are currently shown.

For example, after the flop, with community cards of:

```
AS 5C 4D
```

The nuts would be a 2 and a 3 of any suit, making a straight.

Of course, when the next community card is shown, the nuts may change. Suppose another 4 is shown.

```
AS 5C 4D 4H
```

Now, the nuts would be 4S and 4C making 4 of a kind.

Write a function named **Nuts** which takes as its right argument a character vector representing the shown community cards and returns a vector of character vectors representing the nuts. If the suit of a card does not matter, use * for the suit. It's possible that there may only be one card, or none, that will make the nuts.

```

      Nuts 'AS 5C 4D'
2* 3*
      Nuts 'AS 5C 4D 4H'
4S 4C
      Nuts 'AS KS QS TS'
JS
      Nuts 'AS KS QD JH TC'  A empty vector is returned

```

Task 5 – What are the Odds?

Write a function named **Odds** that takes as its left argument a character vector representing the shown community cards and as its right argument a vector of character vectors representing the players' hole cards. The result of Odds is a numeric vector of length (1 + the number of players) containing values between 0 and 1. The first value is the percent probability of a tie between all players and the remaining values are the percent probability that the corresponding hand will win. The sum of the result should equal 1.

Example:

Note - numbers have been rounded to 4 decimal places for this example, you should not round your results

```

  ⌈←r ← 'TD AC JH' Odds 'QS 6S' '5H 5C' 'TS 5D'
0.0133 0.3101 0.0399 0.6367
  +/r

```

1

The example shows a 1.33% probability of a tie between all three hands, a 31.01% 3.99% and 63.67% winning probabilities for each corresponding hand.




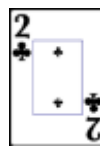



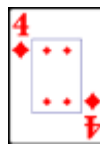

Basic Texas Hold'em Play

Play begins with each player being dealt two cards face down. (As in most poker games, the deck is a standard 52-card deck containing no jokers.) These cards are the player's *hole* cards. These are the only cards each player will receive individually, and they will only (possibly) be revealed at the showdown.

The hand begins with a "pre-flop" betting round. We won't go into the details of betting as they are not pertinent to the problem. After the pre-flop betting round, assuming there remain at least two players taking part in the hand, the dealer deals a flop, three face-up community cards. The flop is followed by a second betting round. After the flop betting round ends, a single community card (called the turn) is dealt, followed by a third betting round. A final single community card (called the river) is then dealt, followed by a fourth betting round and the showdown, if necessary.

If a player bets and all other players fold, then the remaining player is awarded the pot and is not required to show his hole cards. If two or more players remain after the final betting round, a showdown occurs. On the showdown, each player plays the best poker hand they can make from the seven cards comprising his two hole cards and the five community cards. A player may use both of his own two hole cards, only one, or none at all, to form his final five-card hand. If the five community cards form the player's best hand, then the player is said to be *playing the board* and can only hope to split the pot, because each other player can also use the same five cards to construct the same hand.

If the best hand is shared by more than one player, then the pot is split equally among them. It is common for players to have closely valued, but not identically ranked hands. Nevertheless, one must be careful in determining the best hand; if the hand involves fewer than five cards, (such as two pair or three of a kind), then kickers are used to settle ties (see the example below). Note that the card's *numerical* rank is of sole importance; suit values are irrelevant in Hold'em.

Hand 1					
Hand 2					

In the above example, Hand 2 beats Hand 1 because although they each have a pair of Queens, Hand 2 has a higher kicker ($J\heartsuit$ versus $9\heartsuit$)

Ranks of Poker Hands From High to Low

Hand Type	Description	Example(s)
Straight Flush	5 cards in consecutive rank all of the same suit. An Ace-high straight flush is called a "Royal Flush" and is the highest possible hand. Aces may be treated as high or low. If more than one player has a straight flush, the highest ranking hand wins.	AD KD QD JD TD 5C 4C 3C 2C AC
Four of a Kind	4 cards of the same rank. If more than one player has 4 of a kind, the player with highest ranking set of 4 wins.	7S 7H 7D 7C 9S
Full House	3 of one rank and 2 of another. If more than one player has a full house, the player with the highest ranking set of 3 wins.	TH TD TC KS KD
Flush	All 5 cards of the same suit. If more than one player has a flush, the player with the highest rank dissimilar card wins (see example).	KD QD TD 6D 4D beats KS QS 8S 7S 6D
Straight	5 cards of consecutive rank of more than one suit. Aces may be treated as high or low. If more than one person has a straight, the player with the highest ranking straight win.	AS KD QS JH TC 5D 4D 3S 2C AH
Three of a Kind	3 cards of the same rank and 2 cards of unequal rank. If more than one player has 3 of a kind, the player with the higher ranking set of 3 wins.	9S 9H 9C AD JS
Two Pair	2 cards of one rank and 2 cards of a different rank. If more than one player has two pair, the player with the highest ranking pair wins. If two players both have the same rank pairs, the rank of the fifth card (the kicker) determines the winner.	QH QD 7D 7C AS
One Pair	2 cards of one rank. If two or more players have one pair, the player with the highest ranking pair wins. If two players have the same rank pair, the player with the higher rank dissimilar kicker wins.	6D 6C KS QD 7H
High Card	If no player has one of the hand types listed above, the ranks of the cards from high to low are compared and the player with the highest ranking hand wins.	JH 9D 5C 4S 2D

Problem 2 – When in Rome...

Background

Roman numerals, as used today, are based on seven symbols:

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1,000

Numbers are formed by combining symbols together and adding the values. For example, MMVI is $1000 + 1000 + 5 + 1 = 2006$. Generally, symbols are placed in order of value, starting with the largest values. When smaller values precede larger values, the smaller values are subtracted from the larger values, and the result is added to the total. For example $MCMXLIV = 1000 + (1000 - 100) + (50 - 10) + (5 - 1) = 1944$.

There has never been a universally accepted set of rules for Roman numerals. Because of this lack of standardization, there may be multiple ways of representing the same number in Roman numerals. Despite the lack of standardization, an additional set of rules has been frequently applied for the last few hundred years.

- The symbols "I", "X", "C", and "M" can be repeated three times in succession, but no more. (They may appear four times if the third and fourth are separated by a smaller value, such as XXXIX.) "D", "L", and "V" can never be repeated.
- "I" can be subtracted from "V" and "X" only. "X" can be subtracted from "L" and "C" only. "C" can be subtracted from "D" and "M" only. "V", "L", and "D" can never be subtracted.
- Only one small-value symbol may be subtracted from any large-value symbol.
- A number written in Arabic numerals can be broken into digits. For example, 1903 is composed of 1, 9, 0, and 3. To write the Roman numeral, each of the non-zero digits should be treated separately. In the above example, 1,000 = M, 900 = CM, and 3 = III. Therefore, 1903 = MCMIII.

Using this additional set of rules, there is only one possible Roman numeral for any given number.

In addition, for this problem, we will add the following rules.

- 0 (zero) should be represented by an empty character vector.
- Negative numbers should be preceded by an APL high minus (⊖)
- Non-integers should be rounded up (.5 and above rounds up).
- Larger numbers simply have a number of leading M's. e.g. 5005 is represented at "MMMMMV".

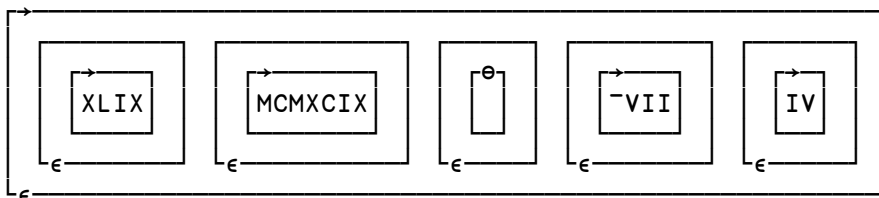
Task 1 – ... Do as the Romans Do

Write a function named **AtoR** that converts an array of Arabic numbers to its Roman equivalent based on the rules above. Each Roman number should be a scalar and the shape of the result should be the same as the shape of the right argument.

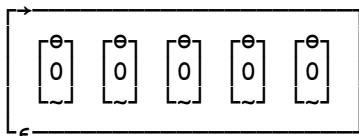
```
AtoR 1903
MCMIII
```

```
v ← 49 1999 0 7 4.3
AtoR v
XLIX MCMXCIX 7 VII IV
```

`]display AtoR v A` `]display` is a user command to display structure



`]display ρ¨ AtoR v A` show that each element of the result is a scalar



`(ρ¨v)≡ρ¨ AtoR v A` verify that result shape is identical to argument

Task 2 – And Back Again

Write a function named **RtoA** that converts an array of Roman numerals into Arabic. As with **AtoR**, the shape of the result should match the shape of the argument, except that a single Roman number need not be enclosed as a scalar.

```
RtoA AtoR v
49 1999 0 7 4
```

```
RtoA 'MCMLXI'
1961
```

```
ρRtoA 'MCMLXI' A verify the result is a scalar (rank 0)
0
```

Task 3 – Smooth Operator

One of the neat features of APL is its use of operators, functions that take other functions (or data) as arguments and return derived functions as their result. For instance, there's the outer product operator (`∘.`) which takes a function and applies it to all combinations of the left and right arguments.


```

      2 3 4 0.× 2 4 6 8
4  8 12 16
6 12 18 24
8 16 24 32

```

Similarly, the reduction operator "reduces" the rank of an array by applying its function between elements along an axis of the data.

```

      table←2 4ρ8?100 A make a 2 row, 4 column matrix of 8 numbers between 1 and 100

      table
14 76 46 54
22  5 68 94

      +/[2]table A row sums - the second dimension (columns) gets "reduced"
190 189

      +/table A if no axis is specified, reduction is performed on the last dimension
190 189

      +/[1]table A column sums - reduction along the first dimension (rows)
36 81 114 148

      +/table A / indicates reduction on the first dimension
36 81 114 148

      [/table A row maximum - reduction can take any function appropriate
                                for the domain of the data
76 94

      [/table A row minimum
14 5

```

In addition to APL's built-in operators, you can write user defined operators! So, as you've probably guessed, your task is to write a user defined operator named **Roman** to perform Roman math.

```

      'III'+Roman'II'
V

      ιRoman'X'
I II III IV V VI VII VIII IX X

      +/RomanιRoman'X'
LV

```

Don't worry about "mixed" data types. We don't expect this to work...

```
'II' 'III'ρRoman ι6
```

But the following should...

```

      'II' 'III'ρRoman ιRoman'VI'
I  II  III
IV V  VI

      'II' 'III' {αριω}Roman 'VI'
I  II  III
IV V  VI

```

Problem 3 – Digital Fun

Background

There have been many recreational puzzles that make use of the digits 123456789. Some have involved inserting mathematical signs between the digits in order such that the resulting expression equals a particular number, 100 for instance. $100 = 1+2+3+4+5+6+7+(8\times 9)$. This problem adds an APL twist.

In the Problem3 namespace, we have provided you with a function named **Problem3**, a sample of which is below.

```
▽ r←Problem3
[1] r←100ρ0
[2] r[1]←123456789
[3] r[2]←123456789
...
[101] r[100]←123456789
▽
```

Your task is to edit each line by inserting any APL symbols other than 0-9 before, between, or after the digits 123456789 and to the right of the assignment arrow (\leftarrow) to make the N^{th} element of result **r** equal N.

Your goal is to do this with the fewest possible characters. For example, the first result element can be accomplished with the insertion of a single character.

```
r[1]←×123456789    or
r[1]←1[23456789
```

When complete **Problem3**≡ι100 should return a 1.

The judges will use $\rho>, / \square \text{NR}$ 'Problem3' to determine the size of your code, so this is one case where we don't want you to comment your code!

Problem 4 – Nurikabe

Nikoli (にこり, *nikori*) **Co., Ltd.** is a Japanese publisher that specializes in games and, especially, logic puzzles. *Nikoli* is also the nickname of a quarterly magazine (whose full name is *Puzzle Communication Nikoli*) issued by the company.¹ While Sudoku may be the best known puzzle from *Nikoli*, they have published scores of other fascinating games. One of these is Nurikabe, a description of which may be found at <http://www.nikoli.com/en/puzzles/nurikabe/>.

The **nurikabe** (ぬりかべ) is a *Yōkai*, or spirit, from Japanese folklore. It manifests as a wall that impedes or misdirects walking travelers at night. Trying to go around is futile as it extends itself forever. Knocking on the lower part of the wall makes it disappear.²

For the purposes of this problem, a Nurikabe puzzle can be represented as an integer matrix with 0 (zero) in empty cells. For instance, the sample puzzle found at <http://www.nikoli.com/en/puzzles/nurikabe/rule.html>,

	3				1	
2			1			
	1			2		
		2				
1				1		6

would be represented by:

```

7 7 0 3 0 0 0 1, (8/0), 2 0 0 1, (11/0), 1 0 0 2, (4/0), 2, (4/0), 1 0 0 0 1 0 6
0 3 0 0 0 1 0
0 0 0 0 0 0 0
2 0 0 1 0 0 0
0 0 0 0 0 0 0
0 1 0 0 2 0 0
0 0 2 0 0 0 0
1 0 0 0 1 0 6

```

A solved puzzle is represented in the same manner except that there is a $\bar{1}$ in the place of any filled in square. The solution to the sample puzzle above

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

would be represented as:

¹ Source: Wikipedia

```

-1 3 0 0 -1 1 -1
-1 -1 -1 -1 -1 -1 -1
 2 0 -1 1 -1 0 0
-1 -1 -1 -1 -1 -1 0
-1 1 -1 0 2 -1 0
-1 -1 2 -1 -1 -1 0
 1 -1 0 -1 1 -1 6

```

Task 1 – Seeking Validation

Your first task is to write a function named **Valid** which takes as its right argument an integer matrix representing a possible Nurikabe puzzle solution and returns a 1 if the argument is a valid Nurikabe solution and 0 otherwise.

If passed the solution matrix shown above **Valid** should return a 1.

Other simple examples include:

```

Valid 1 1p1
1
Valid 2 2p-1 -1 -1 1
1
Valid 2 2p-1 -1 2 0
1
Valid 2 2p1 -1 -1 1
0

```

Task 2 – Solved!

Your final task for this year's contest is to write a function named **Nurikabe** which takes as its right argument an integer matrix representing a Nurikabe puzzle as described above and returns as its result an integer matrix representing the solution to the puzzle, also as described above. The judges will test your solution with several different Nurikabe puzzles.

```

p←7 7p0 3 0 0 0 1,(8/0),2 0 0 1,(11/0),1 0 0 2,(4/0),2,(4/0),1 0 0 0 1 0 6
Nurikabe p
-1 3 0 0 -1 1 -1
-1 -1 -1 -1 -1 -1 -1
 2 0 -1 1 -1 0 0
-1 -1 -1 -1 -1 -1 0
-1 1 -1 0 2 -1 0
-1 -1 2 -1 -1 -1 0
 1 -1 0 -1 1 -1 6

Valid Nurikabe p
1

```