APL Programming Guide

First Edition (May 1978)

A form for readers' comments has been provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Marketing Publications, Dept. 824 , 1133 Westchester Avenue, White Plains, New York 10604. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

APL PROGRAMMING GUIDE

VECTOR OPERATIONS

FIRST EDITION

EDITED BY

TIMOTHY P. HOLLS

ABSTRACT

THIS GUIDE SUMMARIZES KEY CONCEPTS, CODING TECHNIQUES, IDIOMS, GUIDELINES, AND TRADE-OFFS WHICH WILL HELP THE APL PROGRAMMER TO PRODUCE EFFICIENT APL CODE.

KEY WORDS: APL, APLSV, PROGRAMMING TECHNIQUES, APL/CMS, VS APL, 5100 *APL*

A PUBLICATION OF DPD SCIENTIFIC MARKETING

APL PROGRAMMING GUIDE

PREFACE

THIS GUIDE IS A COMPILATION OF KEY CONCEPTS, CODING TECHNIQUES, IDIOMS, GUIDELINES, AND TRADE-OFFS IN APL PROGRAMMING. IT IS INTENDED FOR INTERMEDIATE AND ADVANCED APL PROGRAMMERS WHO ARE FAMILIAR WITH APL SYNTAX AND OPERATIONS AS FOUND IN THE APL LANGUAGE MANUALS AND SIMILAR PUBLICATIONS. ITS GOAL IS TO SHOW GOOD IMPLEMENTATIONS OF BASIC APL DATA PROCESSING OPERATIONS.

THIS VOLUME COVERS VECTOR OPERATIONS TOPICS IN APL PROGRAMMING. IT IS DESIGNED TO BE A REFERENCE MANUAL. THE INFORMATION INCLUDED IS CATEGORIZED ACCORDING TO THE TYPE OF OPERATION PERFORMED AND THE DATA BEING PROCESSED. THE TABLE OF CONTENTS IS INTENDED TO SERVE AS THE PRIMARY MEANS FOR LOCATING THE APL CODE FOR A SPECIFIC OPERATION.

THIS GUIDE IS BASED ON EXPERIENCE WITH IBM APLSV SYSTEMS. WHERE APPROPRIATE, THE TECHNIQUES HAVE BEEN CODED SO AS NOT TO BE DEPENDENT ON ANY APL SYSTEM IMPLEMENTATION. HOWEVER, THE READER MUST DETERMINE WHETHER OR NOT CODE CHANGES ARE NECESSARY FOR EXECUTION ON ANOTHER APL SYSTEM. CERTAIN CODE IMPLEMENTATIONS MAY HAVE TO BE SPLIT INTO ADDITIONAL STATEMENTS IF THE ORDER OF EXECUTION OF OTHER APL SYSTEMS REQUIRES THIS.

IT IS THE PHILOSOPHY OF THIS GUIDE TO SHOW ONLY THE BASIC CODE NECESSARY TO PERFORM A GIVEN OPERATION. ONLY A FEW TECHNIQUES ARE ILLUSTRATED AS COMPLETELY-DEFINED APL FUNCTIONS. IT IS LEFT TO THE READER TO COMBINE THESE BASIC COMPONENTS INTO STANDARDIZED SUBROUTINES OR LARGER FUNCTIONS AS DEEMED APPROPRIATE FOR AN APPLICATION.

* * * * *

MORE THAN ONE IMPLEMENTATION OF A TECHNIQUE IS SHOWN WHEN THERE ARE SIGNIFICANT DIFFERENCES IN THE NATURE OF THE IMPLEMENTATIONS. EACH IMPLEMENTATION IS THE "BEST" ONE FOR ITS PARTICULAR APPROACH. THE ORDER IN WHICH THE IMPLEMENTATIONS ARE LISTED FOR EACH TECHNIQUE OR EACH TRADE-OFF IS BASED ON MEASUREMENTS OF THE TIME TAKEN TO PROCESS A VECTOR OF ABOUT 25 ELEMENTS. THE FASTEST IMPLEMENTATION IS SHOWN FIRST.

THE READER IS CAUTIONED, HOWEVER, THAT SUCH CHARACTERISTICS AS EXECUTION SPEED, STORAGE REQUIRED, DATA REPRESENTATION AND HANDLING, ETC., ARE DEPENDENT ON THE APL SYSTEM IMPLEMENTATION. THESE CHARACTERISTICS MAY ALSO CHANGE AS AN APL SYSTEM IS MODIFIED AND IMPROVED. THEREFORE NO EXPLICIT MEASUREMENTS OF THESE CHARACTERISTICS HAVE BEEN INCLUDED IN THIS GUIDE. READERS WHO CONSIDER THESE CHARACTERISTICS IMPORTANT IN THEIR APPLICATIONS SHOULD DETERMINE THEM FOR EACH TECHNIQUE ON THE APL SYSTEM WHICH THEY ARE USING.

PREFACE (CONTD)

EXECUTION SPEED CAN ALSO VARY WITH THE LENGTH AND NATURE OF THE VARIABLES PROCESSED. AN IMPLEMENTATION WHICH IS FASTER FOR SMALL VARIABLES MAY BE SLOWER FOR LARGE ONES. SIMILARLY, LOGICAL DATA (BITS) MAY BE HANDLED MORE SLOWLY THAN BYTE OR INTEGER DATA.

* * * * *

THE IMPLEMENTATIONS SHOWN IN THIS GUIDE ARE LARGELY NON-LOOPING SOLUTIONS USING THE POWER OF APL PRIMITIVE FUNCTIONS. THE READER SHOULD BE AWARE, HOWEVER, THAT IMPLEMENTATIONS WITH LOOPING CAN BE FASTER AND MORE EFFICIENT FOR LARGE DATA VARIABLES IN CERTAIN SITUATIONS. LOOPING WILL ALSO HELP AVOID "WS FULL" INTERRUPTIONS, ALLOWING LARGER VARIABLES TO BE PROCESSED.

SOME ALTERNATE IMPLEMENTATIONS WHICH ARE KNOWN TO BE POORER HAVE BEEN INCLUDED IN ORDER TO DOCUMENT THEIR EXISTENCE AS VARIATIONS. CERTAIN ONES FOUND IN OLDER APL FUNCTIONS REPRESENT EARLIER SOLUTIONS DEVELOPED BEFORE NEW APL PRIMITIVE FUNCTIONS WERE ADDED TO THE LANGUAGE. SOME OF THE ALTERNATE IMPLEMENTATIONS, WHILE LESS EFFICIENT FOR PERFORMING THE ENTIRE TECHNIQUE, MAY NEVERTHELESS BE BETTER IN A SPECIFIC APPLICATION WHERE THEIR INTERMEDIATE RESULTS ARE ALSO USEFUL OR ALREADY AVAILABLE FROM PREVIOUS PROCESSING. THE FINAL RESULT MIGHT BE OBTAINED FASTER BY STARTING WITH THE INTERMEDIATE DATA THAN BY PROCESSING FROM SCRATCH WITH THE FASTEST TECHNIQUE. MINOR VARIATIONS IN IMPLEMENTATION, E.G., COMPLEMENTARY OPERATIONS, ARE GENERALLY OMITTED. THE NOTATION "↔" MEANS "IS THE SAME AS".

EXPLANATIONS OF THE DETAIL-PROCESSING OCCURRING IN THESE TECHNIQUES ARE INCLUDED ONLY WHERE THEY WERE CONSIDERED NECESSARY. READERS WHO SEEK A SPECIFIC TECHNIQUE MAY SIMPLY COPY THE CODE AND VERIFY THE DESIRED OPERATION AND RESULTS. THOSE WHO WANT TO INCREASE THEIR GRASP AND UNDERSTANDING OF APL PROCESSING ARE ENCOURAGED TO STUDY THE CODE IN DETAIL. THIS CAN CONSIST OF FIRST WORKING IT OUT ON PAPER AND THEN EXECUTING EACH PORTION IN SEQUENCE TO EXAMINE THE RESULTS. THIS EXERCISE PROVES TO BE A VERY GOOD WAY TO UNDERSTAND AND MASTER APL SEMANTICS AND PROGRAMMING LOGIC. IT CONTRIBUTES TO THE ABILITY TO RECOGNIZE ENTIRE TECHNIQUES OR "IDIOMS" OF APL WHEN SUBSEQUENTLY READING APL PROGRAMS. SOME OF THESE TECHNIQUES ARE THEMSELVES COMBINATIONS OF SEVERAL SIMPLER TECHNIQUES.

CERTAIN BRIEF APL OPERATIONS OCCUR REPEATEDLY IN THESE TECHNIQUES. THEY CAN BE THOUGHT OF AS "KERNEL" APL OPERATIONS AND ARE IDENTIFIED AS SUCH IN THE TEXT. WHILE THEY ARE NOT NECESSARILY USED TO PRODUCE FINAL RESULTS IN THEMSELVES, THEY FORM THE BASIS FOR MANY OTHER MORE COMPLEX TECHNIQUES. THEREFORE THEY ARE WORTH RECOGNIZING AND LEARNING.

PREFACE (CONTD)

MANY OPERATIONS ARE SHOWN WITH A GENERALIZED TEST VARIABLE, E.G., "Q", AND THEY MAY BE USABLE FOR EITHER NUMERIC OR CHARACTER DATA. IN APPLICATIONS WHERE ONLY ONE KIND OF DATA IS PROCESSED, THE SPECIFIC DATA ELEMENTS, E.G., 0 OR ' ', MAY BE SUBSTITUTED IN THE EXPRESSION RATHER THAN ASSIGNING THEM TO THE VARIABLE.

WHEREVER POSSIBLE AND APPROPRIATE, THE IMPLEMENTATIONS OF THESE TECHNIQUES ARE CODED TO EXECUTE THE SAME REGARDLESS OF THE VALUE OF THE INDEX ORIGIN ⎕IO. THIS IS DONE TO INCREASE THE USABILITY OF THE TECHNIQUES. HOWEVER IT FREQUENTLY MAKES THE APL CODE MORE DIFFICULT TO READ AND MORE CUMBERSOME. IF A TECHNIQUE IS TO BE EXECUTED IN AN ENVIRONMENT WITH AN UNCHANGING INDEX ORIGIN, IT IS SUGGESTED THAT THE READER SUBSTITUTE THE ORIGIN VALUE FOR ⎕IO WHERE IT APPEARS IN THE CODE AND THEN SIMPLIFY THE APL EXPRESSIONS WHICH RESULT.

* * * * *

THE PAGES OF THIS GUIDE HAVE BEEN FORMATTED AND GENERATED BY EXECUTION OF APL FUNCTIONS. IN ALMOST ALL CASES THE RESULTS SHOWN FOR EACH OPERATION HAVE BEEN GENERATED BY ACTUALLY EXECUTING THE ILLUSTRATED APL CODE. THE CODE FOR ALTERNATE IMPLEMENTATIONS HAS BEEN EXECUTED AND THE RESULTS COMPARED WITH THE MAIN ILLUSTRATION. CODE WHICH IS AFFECTED BY THE INDEX ORIGIN HAS BEEN EXECUTED IN BOTH ORIGINS AND THE RESULTS COMPARED (WITH ANY NECESSARY ADJUSTMENTS). THEREFORE THE CORRECTNESS OF THE CODE SHOWN FOR EACH TECHNIQUE HAS BEEN VERIFIED IN THE PROCESS OF GENERATING THESE PAGES.

READERS WHO KNOW OF OTHER "BETTER" IMPLEMENTATIONS OF ANY TECHNIQUE OR WHO DETERMINE IMPROVEMENTS TO ANY IMPLEMENTATIONS SHOWN IN THIS GUIDE ARE REQUESTED TO SEND THIS INFORMATION TO THE EDITOR. CONTRIBUTIONS AND SUGGESTIONS FOR ADDITIONAL TECHNIQUES ARE ALSO WELCOME.

THE EDITOR IS INDEBTED TO MANY APL PROGRAMMERS, BOTH KNOWN AND UNKNOWN, FOR THE CODE WHICH HAS BEEN INCLUDED IN THIS GUIDE. SPECIAL APPRECIATION IS EXTENDED TO THE REVIEWERS, WHO CONTRIBUTED IMPROVEMENTS TO THE CODE, THE TEXT, AND THE FORMAT OF THIS GUIDE.

CERTAIN MATERIAL IDENTIFIED BY FOOTNOTES HAS BEEN ADAPTED FROM THE PUBLICATIONS LISTED BELOW. SOME OF THE MATERIAL IS UNCHANGED EXCEPT FOR VARIABLE NAMES AND APL SYSTEM DIFFERENCES.

1. PERLIS, ALAN J., AND RUGABER, SPENCER, THE APL IDIOM LIST, RESEARCH REPORT 87, YALE UNIVERSITY DEPARTMENT OF COMPUTER SCIENCE, APRIL 1977.

PREFACE (*CONTD*)

2. *SMITH, ROBERT A.*, ***BOOLEAN FUNCTIONS AND TECHNIQUES***, *WORKING MEMORANDUM NO.* 106, *SCIENTIFIC TIME SHARING CORPORATION, FEBRUARY* 1975, *REVISED JULY* 1975.

3. *SYKES, ROY A.*, ***USE AND MISUSE OF APL - EFFICIENT CODING TECHNIQUES***, *SHARE XL, MARCH* 7, 1973, *SCIENTIFIC TIME SHARING CORPORATION*.

VECTOR OPERATIONS -- TABLE OF CONTENTS

VECTOR OPERATIONS -- TABLE OF CONTENTS (CONTD)

VECTOR OPERATIONS -- TABLE OF CONTENTS (CONTD)

CREATING A VECTOR V

BY CATENATION

TYPE 1: *CATENATING INDIVIDUAL ELEMENTS*

```
⍝ INITIALIZE TO AN "EMPTY" VECTOR

      V←⍳0       OR      V←''       OR      V←0⍴...

⍝ ADD ELEMENTS BY CATENATION

      V←V,45 78 23

⍝ USE THIS METHOD WHEN THE FINAL SIZE OF V IS UNKNOWN

⍝ CAUTION:  CATENATION REQUIRES THAT ⎕WA > THE STORAGE
⍝   TAKEN UP BY "V".  WHEN ⎕WA BECOMES <, A "WS FULL"
⍝   WILL OCCUR.
```

TYPE 2: *CATENATING SEVERAL VECTORS*

```
⍝ GENERATE SEPARATE "PARTIAL"-VECTORS

      T←'FINAL SUMMARY REPORT OF ANNUAL SALES'
      D←'DECEMBER 31, 19', 2 0 ⍕Y
      BR←'BRANCH OFFICE NO. ',⍕N

⍝ CATENATE VECTORS TO FORM LONGER VECTOR

      HDR←T,' -- ',BR,' -- ',D

⍝ USE THIS METHOD WHEN FINAL VECTOR WOULD BE TOO LONG
⍝  FOR ONE FUNCTION STATEMENT.  THIS ALSO ALLOWS THE
⍝  THE INDIVIDUAL VECTORS TO BE USED IN MORE THAN ONE
⍝  STATEMENT.
```

(CONTD ON NEXT PAGE)

CREATING A VECTOR V

BY CATENATION (*CONTD*)

TYPE 3: *MANUALLY ENTERING A VECTOR WHERE THE DATA EXCEEDS THE TERMINAL INPUT LINE LENGTH*

```
⍝ CATENATE A "⎕" AT THE END OF EACH LINE WHICH IS
⍝  TO BE CONTINUED

        V←2.3  9.7  6.2  5.8  ...  43.79, ⎕
 ⎕:
        93.7  105.2  125.3  ...  155.1, ⎕
 ⎕:
        ... ETC.

⍝ CHARACTER INPUT MUST BE TYPED WITHIN QUOTES
```

CREATING A VECTOR V

CATENATING UNIQUE ELEMENTS Q TO V AND INCREMENTING COUNTERS C

```
⍝ CATENATE THE ELEMENTS ONLY IF THEY DON'T OCCUR IN V

        V←V,(L←~Q∊V)/Q          **

⍝ INCREMENT THE CORRESPONDING COUNTER IN C FOR EACH
⍝  ELEMENT

        C←(C,L/0)+V∊Q           **

⍝ ⍴V ←→ ⍴C
⍝ C[I] IS THE COUNT OF OCCURRENCES OF V[I]
```

EXAMPLE:
```
V←'MECRVKID'
C←2 5 2 1 1 2 3 1
Q←'BASED'
```

RESULT:
```
V:  MECRVKIDBAS
C:  2 6 2 1 1 2 3 2 1 1 1
```

```
⍝ IF ANY ELEMENTS IN Q OCCUR MORE THAN ONCE:

        V←V,(L←~U∊V)/U←((⍳⍴Q)=Q⍳Q)/Q

        C←(C,L/0)++/V∘.=Q
```

EXAMPLE:
```
Q←'MISSISSIPPI IS THE MAGNOLIA STATE'
```

RESULT:
```
V:  MECRVKIDBASP THGNOL
C:  4 8 2 1 1 2 9 2 1 4 7 2 4 3 1 1 1 1 1
```

** ADAPTED FROM (1) *THE APL IDIOM LIST*

CREATING A VECTOR V

BY ASSIGNMENT AND INDEXING

```
⍝ FIRST ASSIGN ¨N¨ FILL-ELEMENTS TO THE VECTOR, E.G.,
⍝  PAD-ELEMENTS (0 OR BLANK);  N = A POSITIVE INTEGER

       V←Nρ0

⍝ NOTE:  Nρ0  PRODUCES 1-BIT ELEMENTS.  THESE WILL
⍝   INCREASE TO INTEGER OR FLOATING POINT ELEMENTS
⍝   WHEN THE FIRST VALUE ≠ 0 OR 1 IS ASSIGNED.

⍝ INITIALIZE INDEXING VARIABLE

       PTR←0            [⎕IO←1]

⍝ STORE ELEMENTS IN VECTOR BY INDEXING, CHECKING FOR
⍝  END OF VECTOR

       →(N<PTR←PTR+1)/END
       V[PTR]←45

⍝ ALTERNATIVELY

       →(N<¯1↑I←PTR+⍳K)/END
       V[I]←1 2 3 ... K          [⎕IO←1]
       PTR←PTR+K


⍝ USE THIS METHOD IF THE FINAL SIZE OF VECTOR (OR AN
⍝  UPPER LIMIT ON THE SIZE) IS KNOWN

⍝ CAUTION:  IF THE STORAGE TAKEN UP BY ¨V¨ > ⎕WA, NO
⍝   OPERATION CAN BE PERFORMED WHICH WOULD DUPLICATE ¨V¨
⍝   (WILL RESULT IN A ¨WS FULL¨)
```

CREATING A VECTOR V

BY RESHAPING ANOTHER VARIABLE W

```
V←NρW
```

```
⍝ W MAY BE AN ARRAY OF ANY RANK (0, 1, 2, 3, ...)
⍝ N MUST BE AN INTEGER ≥ 0
```

EXAMPLE:
```
W←'⎕'
N←10
```

RESULT:
```
⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

EXAMPLE:
```
W←'ABCD'
N←13
```

RESULT:
```
ABCDABCDABCDA
```

EXAMPLE:
```
W←2 3 4ρ 1 0 0 1 0
N←17
```

RESULT:
```
1 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0
```

```
⍝ TO REPLICATE A VECTOR W N-TIMES:
```

```
V←(N×ρW)ρW
```

```
⍝ N MUST BE AN INTEGER ≥ 0
```

EXAMPLE:
```
W←'1234'
N←4
```

RESULT:
```
1234123412341234
```

CREATING A VECTOR V

GENERATING A VECTOR USING A LOGICAL VECTOR AND A SCALAR Q

```
V←( LOGICAL VECTOR )\Q
```

⍝ *PAD-ELEMENTS* (0 *OR BLANK*) *OCCUPY THE* 0-*BIT POSITIONS*

EXAMPLE: *UNDERLINING A HEADING (REPLACE Q WITH* '¯')

```
HDR←'THIS IS A HEADING'
V←(HDR≠' ')\'¯'
```

RESULT:

```
---- -- - -------
```

EXAMPLE:

```
L←1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
V←L\'∘'
```

RESULT:

```
∘∘  ∘∘  ∘∘  ∘∘
```

CREATING A VECTOR V

GENERATING A VECTOR OF INDICES

IN FIELDS OF WIDTHS W

```
V←(+/W)ρ0
V[+\(ι□IO),¯1↓W]←1
V←+\V
```

EXAMPLE: W←2 2 4 1

RESULT: 1 1 2 2 3 3 3 3 4 [□IO←1]

```
⍝ ALTERNATE IMPLEMENTATIONS:

        V←□IO++\(ι+/W)∊□IO++\W

        V←□IO++/(ι+/W)∘.≥□IO++\W

⍝ TO GENERATE A VECTOR OF THE ELEMENTS IN VECTOR "X",
⍝  CODE:  X[V]
```

EXAMPLE: X←'Δ⊂⋆⊃∇'
W←3 1 1 2 3

RESULT: ΔΔΔ⊂⋆⊃⊃∇∇∇

CREATING A VECTOR V

GENERATING A VECTOR OF INDICES

GENERATING THE "X"TH AXIS INDICES I FOR ALL ELEMENTS OF ARRAY A, CORRESPONDING TO RAVEL-A (,A)

```
R←(~⎕IO)+X-⍴⍴A
I←,(R⌽⍳⍴⍴A)⍉(R⌽⍴A)⍴⍳(⍴A)[X]
```

```
⍝ THIS CAN BE SIMPLIFIED FOR ARRAYS OF RANK 1 OR 2:

⍝ RANK 1 (= VECTOR):  I←⍳⍴A

⍝ NOTE:  THIS IS A "KERNEL" APL OPERATION

⍝ RANK 2 (= MATRIX):
⍝      ROW INDICES (X=1):  I←,⍉(⌽⍴A)⍴⍳1↑⍴A
⍝      COL INDICES (X=2):  I←,(⍴A)⍴⍳¯1↑⍴A

⍝ NOTE:  FOR RANK ≥ 3, THE EXPRESSION FOR THE LAST AXIS
⍝    (COLUMNS) IS THE SAME AS SHOWN ABOVE FOR RANK 2.
```

EXAMPLE:

```
A←2 3 2⍴⍳12

 1  2
 3  4
 5  6

 7  8
 9 10
11 12

,A
1 2 3 4 5 6 7 8 9 10 11 12
```

RESULT:

```
CORRESPONDING PLANE INDICES (X←1):  [⎕IO←1]
1 1 1 1 1 1 2 2 2 2 2 2

CORRESPONDING ROW INDICES (X←2):
1 1 2 2 3 3 1 1 2 2 3 3

CORRESPONDING COLUMN INDICES (X←3):
1 2 1 2 1 2 1 2 1 2 1 2
```

(CONTD ON NEXT PAGE)

CREATING A VECTOR V

GENERATING A VECTOR OF INDICES

GENERATING THE "X"TH AXIS INDICES I FOR ALL ELEMENTS OF ARRAY A, CORRESPONDING TO RAVEL-A (,A) (CONTD)

```
⍝ ALTERNATE IMPLEMENTATION:

        I←⎕IO+((⍴A)⊤(⍳×/⍴A)-⎕IO)[X;]

⍝ NOTE:  THIS IMPLEMENTATION GENERATES INDICES FOR ALL
⍝   THE AXES AT ONE TIME
```

CREATING A VECTOR V

GENERATING A VECTOR OF INDICES

IN INCREASING-DECREASING SEQUENCE, WITH LENGTH L

```
V←(⍳⌈H),⌽⍳⌊H←L÷2
```

EXAMPLE: `L←8`

RESULT: `1 2 3 4 4 3 2 1      [⎕IO←1]`

EXAMPLE: `L←13`

RESULT: `0 1 2 3 4 5 6 5 4 3 2 1 0      [⎕IO←0]`

```
⍝ NOTE:  IF H ← A DIMENSION OF A MATRIX M, V CAN BE
⍝    USED TO EXPAND M SYMMETRICALLY, I.E.,  M[V;] FOR
⍝    ROW SYMMETRY,  M[;V] FOR COLUMN SYMMETRY
```

IN INCREASING-DECREASING SEQUENCE, WITH LENGTH L, AND WITH MAXIMUM VALUE K

```
V←K⌊(⍳⌈H),⌽⍳⌊H←L÷2
```

EXAMPLE: `L←12`
`K←3`

RESULT: `1 2 3 3 3 3 3 3 3 3 2 1      [⎕IO←1]`

CREATING A VECTOR V

GENERATING A VECTOR WITH A RANGE OF NUMBERS

SUCCESSIVE INTEGERS FROM J TO K

```
V←J+(×K-J)×(⍳1+|K-J)-⎕IO
```

EXAMPLE:
```
J←7
K←¯2
```

RESULT:
```
7 6 5 4 3 2 1 0 ¯1 ¯2
```

SUCCESSIVE REAL NUMBERS FROM J TO K INCREMENTING BY D

```
V←J+(D××K-J)×(⍳1+|⌊(K-J)÷D)-⎕IO

⍝ J AND K ARE ANY REAL NUMBERS; D REAL AND > 0
```

EXAMPLE:
```
J←1.4
K←8.2
D←.7
```

RESULT:
```
1.4 2.1 2.8 3.5 4.2 4.9 5.6 6.3 7 7.7
```

(*CONTD ON NEXT PAGE*)

CREATING A VECTOR V

GENERATING A VECTOR WITH A RANGE OF NUMBERS (CONTD)

R SUCCESSIVE REAL NUMBERS STARTING WITH N AND INCREMENTING BY D

```
V←N+D×(⍳R)-⎕IO
```

```
⍝ R IS ANY INTEGER ≥ 0; N AND D ARE ANY REAL NUMBERS
```

EXAMPLE:
```
R←10
N←0
D←.1
```

RESULT: 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9

```
⍝ SPECIAL CASE IMPLEMENTATIONS, FOR INTEGER N
⍝  AND D = 1:
```

```
V←(N-⎕IO)+⍳R

V←(N-⎕IO)↓⍳R+N-⎕IO          **
```

EXAMPLE:
```
R←8
N←13
```

RESULT: 13 14 15 16 17 18 19 20

** ADAPTED FROM (3) *USE AND MISUSE OF APL*

CREATING A VECTOR V

GENERATING A LOGICAL (BOOLEAN) VECTOR

WITH K LEADING 1'S AND LENGTH L

```
V←L↑Kρ1
```

EXAMPLE:
```
L←12
K←5
```

RESULT: 1 1 1 1 1 0 0 0 0 0 0 0

```
⍝ ALTERNATE IMPLEMENTATION:

V←(K-~⎕IO)≥⍳L
```

WITH K TRAILING 1'S AND LENGTH L

```
V←(-L)↑Kρ1
```

RESULT: 0 0 0 0 0 0 0 1 1 1 1 1

```
⍝ ALTERNATE IMPLEMENTATION:

V←(L-K+~⎕IO)<⍳L
```

CREATING A VECTOR V

GENERATING A LOGICAL (BOOLEAN) VECTOR

DEFINING THE FIELD PARTITION P OF V

```
⍝ ⍴P ←→ ⍴V
⍝ P[I] = 1 WHERE V[I] IS THE FIRST (OR LAST) ELEMENT
⍝  OF A FIELD; OTHERWISE P[I] = 0

⍝ NOTE:  THIS VECTOR "P" IS REQUIRED WHEN PERFORMING
⍝   CERTAIN OTHER OPERATIONS ON FIELDS OF V
```

WHEN THERE ARE NO FIELD DELIMITERS

TYPE 1: *FOR K FIELDS WITH SAME WIDTH W*

```
FIRST:  P←(K×W)⍴W↑1

LAST:   P←(K×W)⍴(-W)↑1

⍝ NOTE:  THIS IS A "KERNEL" APL OPERATION

EXAMPLE:   K←5
           W←3

RESULT:    FIRST: 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0
           LAST:  0 0 1 0 0 1 0 0 1 0 0 1 0 0 1

⍝ ALTERNATE IMPLEMENTATIONS:

FIRST:  P←(K×W)⍴1,(W-1)⍴0

LAST:   P←(K×W)⍴((W-1)⍴0),1

⍝ NOTE:  THIS FIELD PARTITION CAN BE USED WHEN
⍝   PROCESSING A K×W MATRIX RAVELED AS A VECTOR
```

(*CONTD ON NEXT PAGE*)

CREATING A VECTOR V

GENERATING A LOGICAL (BOOLEAN) VECTOR

DEFINING THE FIELD PARTITION P OF V (CONTD)

WHEN THERE ARE NO FIELD DELIMITERS (CONTD)

TYPE 2: *FROM A VECTOR I OF INDEX NUMBERS OF THE 1ST ELEMENTS OF EACH FIELD*

```
FIRST:  P←(⍳⍴V)∊I

LAST:   P←1⌽(⍳⍴V)∊I

⍝ NOTE:  THIS IS A "KERNEL" APL OPERATION

EXAMPLE:   V←⍳10
           I←1 4 5 9      [⎕IO←1]

RESULT:    FIRST: 1 0 0 1 1 0 0 0 1 0
           LAST:  0 0 1 1 0 0 0 1 0 1

⍝ NOTE:  "P←(⍳⍴V)∊I" IS ALSO VALID FOR A VECTOR V
⍝    WITH FIELD DELIMITERS
```

(*CONTD ON NEXT PAGE*)

CREATING A VECTOR V

GENERATING A LOGICAL (BOOLEAN) VECTOR

DEFINING THE FIELD PARTITION P OF V (CONTD)

WHEN THERE ARE NO FIELD DELIMITERS (CONTD)

TYPE 3: FROM A VECTOR OF FIELD-WIDTHS W, WHERE ρV ←→ +/W

```
FIRST:   P←(ι+/W)∊+\⎕IO,W

LAST:    P←(ι+/W)∊(+\W)-~⎕IO


EXAMPLE:   W←3 1 4 2

RESULT:    FIRST: 1 0 0 1 1 0 0 0 1 0
           LAST:  0 0 1 1 0 0 0 1 0 1


⍝ ALTERNATE IMPLEMENTATIONS:

FIRST:   B←(G×ρW)ρ(G←⌈/W)↑1
         P←(,W∘.>(ιG)-⎕IO)/B

LAST:    B←(G×ρW)ρ(-G←⌈/W)↑1
         P←(,W∘.>⌽(ιG)-⎕IO)/B
```

(CONTD ON NEXT PAGE)

CREATING A VECTOR V

GENERATING A LOGICAL (BOOLEAN) VECTOR

DEFINING THE FIELD PARTITION P OF V (CONTD)

WHEN THERE ARE NO FIELD DELIMITERS (CONTD)

TYPE 4: *FROM A VECTOR OF FIELDS OF IDENTICAL ELEMENTS*

```
FIRST:  P←1,(1↓V)≠¯1↓V

LAST:   P←((1↓V)≠¯1↓V),1
```

⍝ NOTE: *THIS IS A "KERNEL" APL OPERATION*

⍝ NOTE: *MAY HAVE TO SORT V FIRST TO PRODUCE THE FIELDS*

```
EXAMPLE:   V←'BBDDDDCCCA'

RESULT:    FIRST: 1 0 1 0 0 0 1 0 0 1
           LAST:  0 1 0 0 0 1 0 0 1 1
```

⍝ *ALTERNATE IMPLEMENTATIONS:*

```
FIRST:  P←V≠¯1↓(1↑0ρV),V

LAST:   P←V≠1↓V,1↑0ρV
```

⍝ *THESE CAN BE SIMPLIFIED AS FOLLOWS:*

```
        NUMERIC V       CHARACTER V

FIRST:  P←V≠¯1↓0,V      P←V≠¯1↓' ',V

LAST:   P←V≠1↓V,0       P←V≠1↓V,' '
```

(CONTD ON NEXT PAGE)

CREATING A VECTOR V

GENERATING A LOGICAL (BOOLEAN) VECTOR

DEFINING THE FIELD PARTITION P OF V (CONTD)

WHEN THERE ARE NO FIELD DELIMITERS (CONTD)

TYPE 4: (*CONTD*)

```
⍝ CAUTION:  1ST AND LAST ELEMENTS OF V MUST BE
⍝   DIFFERENT TO USE THE FOLLOWING IMPLEMENTATIONS
```

```
FIRST:  P←V≠¯1⌽V

LAST:   P←V≠1⌽V
```

WHEN THE FIELD DELIMITER IS Q

```
FIRST:  P←S>¯1↓0,S←V≠Q

LAST:   P←S>1↓(S←V≠Q),0
```

```
⍝ NOTE:  EMPTY FIELDS ARE ELIMINATED FROM THE PARTITION

EXAMPLE:   V←',ALPHA,PHI,,BETA'
           Q←','

RESULT:    FIRST: 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0
           LAST:  0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1
```

```
⍝ NOTE:  TO DELETE THE FIELD DELIMITERS FROM V AND P
⍝   FOR OTHER PROCESSING, USE:

        V←S/V      OR      V←(V≠Q)/V

        P←S/P      OR      P←(V≠Q)/P
```

CREATING A VECTOR V

GENERATING A LOGICAL (BOOLEAN) VECTOR

IN FIELDS OF WIDTHS W, ALTERNATING 1'*S AND* 0'*S*

TYPE 1: *K FIELDS WITH SAME WIDTH W*

```
V←(K×W)⍴(W×2)↑W⍴1
```

⍝ *NOTE*: *TO START WITH* 0'*S*, *CODE*: "...(-W×2)..."

EXAMPLE:
```
K←5
W←3
```

RESULT: 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1

TYPE 2: *FROM A VECTOR OF FIELD-WIDTHS W,*
WHERE ⍴V ↔ +/W

```
V←≠\(⍳+/W)∊+\⎕IO,W
```

⍝ *NOTE*: *TO START WITH* 0'*S*, *CODE*: ~*V*

EXAMPLE: *W*←5 2 3 1 4

RESULT: 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1

⍝ *NOTE*: *THIS IS A* "≠ *SCAN*" *OPERATION PERFORMED ON A*
⍝ *LOGICAL VECTOR DEFINING THE 1ST ELEMENTS OF THE*
⍝ *FIELD PARTITION OF V (SEE PREVIOUS TECHNIQUE).*

⍝ *ALTERNATE IMPLEMENTATION*:

```
B←(G×⍴W)⍴(G×2)↑(G←⌈/W)⍴1
V←(,W∘.>(⍳G)-⎕IO)/B
```

⍝ *NOTE*: *TO START WITH* 0'*S*, *CODE*: "...(-G×2)..."

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

INSERTING K ELEMENTS Q BEFORE/AFTER EACH ELEMENT OF V

TYPE 1: *INSERTING PAD-ELEMENTS (0 OR BLANK)*

```
AFTER:    V←,V,((ρV),K)ρQ        [Q←0 OR Q←' ']

BEFORE:   V←,(((ρV),K)ρQ),V
```

EXAMPLE: *INSERTING ZEROES*

```
          V←ι5
          K←2
          Q←0
```

```
RESULT:   AFTER:  1 0 0 2 0 0 3 0 0 4 0 0 5 0 0
          BEFORE: 0 0 1 0 0 2 0 0 3 0 0 4 0 0 5
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

AFTER:    V←,((ρV),K+1)↑((ρV),1)ρV

BEFORE:   V←,((ρV),-K+1)↑((ρV),1)ρV

AFTER:    V←(((K+1)×ρV)ρ(K+1)↑1)\V

BEFORE:   V←(((K+1)×ρV)ρ(-K+1)↑1)\V
```

(CONTD ON NEXT PAGE)

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

INSERTING K ELEMENTS Q BEFORE/AFTER EACH ELEMENT OF V (*CONTD*)

TYPE 2: *INSERTING ANY OTHER ELEMENT Q*

```
AFTER:    V←,V,((ρV),K)ρQ

BEFORE:   V←,(((ρV),K)ρQ),V
```

```
EXAMPLE:    V←ι5
            K←2
            Q←10

RESULT:     AFTER:   1 10 10 2 10 10 3 10 10 4 10 10 5 10 10
            BEFORE: 10 10 1 10 10 2 10 10 3 10 10 4 10 10 5
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

AFTER:    V←(B←((K+1)×ρV)ρ(K+1)↑1)\V
          V[(~B)/ιρV]←Q

BEFORE:   V←(B←((K+1)×ρV)ρ(-K+1)↑1)\V
          V[(~B)/ιρV]←Q
```

```
⍝ ALTERNATE IMPLEMENTATION, FOR K = 1

AFTER:    V←,V,[⎕IO+0.5] Q

               ⎕IO←1:   V←,V,[1.5] Q
               ⎕IO←0:   V←,V,[0.5] Q

BEFORE:   V←,Q,[⎕IO+0.5] V
```

```
⍝ NOTE:  AXIS-NUMBER DECIMAL-PART IS A DECIMAL VALUE
⍝   BETWEEN 0 AND 1;   "0.5" SHOWN HERE
```

```
EXAMPLE:    INSERTING ASTERISKS

            V←'ABCDEFG'
            Q←'*'

RESULT:     AFTER:   A*B*C*D*E*F*G*
            BEFORE:  *A*B*C*D*E*F*G
```

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

INSERTING K ELEMENTS Q BEFORE/AFTER THE ELEMENTS WITH INDICES I

TYPE 1: *INSERTING PAD-ELEMENTS* (0 *OR BLANK*)

```
AFTER:    N←K×ρI←,I                     [Q←0 OR Q←' ']
          V←(V,NρQ)[⍋(ιρV),NρI]

BEFORE:   N←K×ρI←,I
          V←((NρQ),V)[⍋(NρI),ιρV]
```

```
⍝ NOTE:   TO INSERT AFTER FIELDS OF WIDTHS ¨W¨, SET I←+\W

EXAMPLE:    V←'NOWISTHETIME'
            I←3 5 8      [⎕IO←1]
            K←2
            Q←' '

RESULT:     AFTER:  NOW  IS  THE  TIME
            BEFORE: NO  WI  STH  ETIME
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

AFTER:    I←I[⍋I←,I]
          B←((K×N←ρI)+ρV)ρ1
          B[(,⍉(K,N)ρI+~⎕IO)+ιK×N]←0
          V←B\V

BEFORE:   REPLACE ¨~⎕IO¨ WITH ¨-⎕IO¨


AFTER:    I←I[⍋I←,I]
          J←(,⍉(K,N)ρI+~⎕IO)+ιK×N←ρI
          V←(~(ι(K×N)+ρV)∊J)\V

BEFORE:   REPLACE ¨~⎕IO¨ WITH ¨-⎕IO¨


AFTER:    C←((K+1)×ρV)ρL←(K+1)↑1
          V←((,((ιρV)∊I)∘.∨L)/C)\V

BEFORE:   REPLACE ¨L←(K+1)...¨ WITH ¨L←(-K+1)...¨
```

(*CONTD ON NEXT PAGE*)

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

INSERTING K ELEMENTS Q BEFORE/AFTER THE ELEMENTS WITH INDICES I (CONTD)

TYPE 1: (CONTD)

```
⍝ FOR K = 1, THESE SIMPLIFY TO:

AFTER:   V←(V,(⍴,I)⍴Q)[⍋(⍳⍴V),(⍴,I)⍴I]

AFTER:   I←I[⍋I←,I]
         B←((N←⍴I)+⍴V)⍴1
         B[(I+~⎕IO)+⍳N]←0
         V←B\V

AFTER:   I←I[⍋I←,I]
         V←(~(⍳N+⍴V)∊(I+~⎕IO)+⍳N←⍴I)\V
```

TYPE 2: INSERTING ANY OTHER ELEMENT Q

```
AFTER:   N←K×⍴I←,I
         V←(V,N⍴Q)[⍋(⍳⍴V),N⍴I]

BEFORE:  N←K×⍴I←,I
         V←((N⍴Q),V)[⍋(N⍴I),⍳⍴V]
```

EXAMPLE:

```
Q←'*'
K←2
```

RESULT:

```
AFTER:   NOW**IS**THE**TIME
BEFORE:  NO**WI**STH**ETIME
```

```
⍝ ALTERNATE IMPLEMENTATION:

AFTER:   I←I[⍋I←,I]
         B←((K×N←⍴I)+⍴V)⍴1
         B[(,⍉(K,N)⍴I+~⎕IO)+⍳K×N]←0
         V←B\V
         V[(~B)/⍳⍴V]←Q

BEFORE:  REPLACE "~⎕IO" WITH "-⎕IO"
```

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

INSERTING ELEMENT Q IN WIDTHS W BEFORE/AFTER THE ELEMENTS WITH INDICES I

TYPE 1: *INSERTING PAD-ELEMENTS* (0 *OR BLANK*)

```
AFTER:     I←I[S←⍋I←,I]
           C←(G×ρV)ρ(G←1+⌈/W←W[S])↑1
           U←((ιρV)∊I)\W
           V←((,U∘.≥(ιG)-⎕IO)/C)\V

BEFORE:    REPLACE ¨(G←...¨ WITH ¨(-G←...¨ AND
            ¨ιG¨ WITH ¨⌽ιG¨
```

```
⍝ EACH ELEMENT OF W CONTAINS THE NUMBER OF ELEMENTS TO
⍝  BE INSERTED BEFORE/AFTER THE ELEMENT OF V WHOSE
⍝  INDEX IS THE CORRESPONDING ELEMENT OF I;  ρI ←→ ρW
```

```
EXAMPLE:     V←7 6 5 4 3 2 1
             I←2 5 6       [⎕IO←1]
             W←1 3 2

RESULT:      AFTER:  7 6 0 5 4 3 0 0 0 2 0 0 1
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

AFTER:     I←I[S←⍋I←,I]
           K←⎕IO++/(ιρV)∘.>I
           J←(ιρV)+(+\0,W←W[S])[K]
           V←((ι(+/W)+ρV)∊J)\V              **

BEFORE:    REPLACE ¨∘.>¨ WITH ¨∘.≥¨
```

** *ADAPTED FROM* (1) *THE APL IDIOM LIST*

(*CONTD ON NEXT PAGE*)

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

INSERTING ELEMENT Q IN WIDTHS W BEFORE/AFTER THE ELEMENTS WITH INDICES I (*CONTD*)

TYPE 2: *INSERTING ANY OTHER ELEMENT Q*

AFTER:

```
I←I[S←⍋I←,I]
C←(G×⍴V)⍴(G←1+⌈/W←W[S])↑1
U←((⍳⍴V)∊I)\W
V←(B←(,U∘.≥(⍳G)-⎕IO)/C)\V
V[(~B)/⍳⍴V]←Q
```

BEFORE: *REPLACE* "(G←..." *WITH* "(-G←..." *AND* "⍳G" *WITH* "⌽⍳G"

EXAMPLE: Q←¯1

RESULT: 7 6 ¯1 5 4 3 ¯1 ¯1 ¯1 2 ¯1 ¯1 1

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

EXPANDING FIELDS WITH DIFFERENT WIDTHS W TO FIELDS OF THE SAME WIDTH L

RIGHT-JUSTIFIED FIELDS

```
V←(,W∘.>⌽(⍳L)-⎕IO)\V
```

```
⍝ INSERTS PAD-ELEMENTS (0 OR BLANK)
⍝ ¨L¨ MUST BE ≥ ⌈/W
```

EXAMPLE:

```
V←'AABCDABABC'
W←1 4 2 3
L←4
```

RESULT:

```
   AABCD  AB ABC
```

```
⍝ ALTERNATE IMPLEMENTATIONS:
```

```
V←(,(W-~⎕IO)∘.≥⌽⍳L)\V
```

```
V←(,(L-W)∘.≤(⍳L)-⎕IO)\V
```

LEFT-JUSTIFIED FIELDS

```
V←(,W∘.>(⍳L)-⎕IO)\V
```

RESULT:

```
A   ABCDAB  ABC
```

```
⍝ ALTERNATE IMPLEMENTATIONS:
```

```
V←(,W∘.≥(⍳L)+~⎕IO)\V
```

```
V←(,(W-~⎕IO)∘.≥⍳L)\V
```

```
⍝ NOTE:  THESE ARE ¨KERNEL¨ APL OPERATIONS
```

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

INSERTING ELEMENT Q AFTER EVERY "K"TH ELEMENT OF V

```
V←,(((⌈(⍴V)÷K),K)⍴V),Q
```

```
⍝ Q IS A SCALAR;  Q MAY BE A VECTOR ONLY IF ⍴Q ←→ ⌈(⍴V)÷K
```

EXAMPLE:

```
V←'123178'
K←2
Q←'/'
```

RESULT:

```
12/31/78/
```

```
⍝ NOTE:  MAY ALSO CATENATE Q AT THE LEFT, BEFORE THE RAVEL.

⍝        LEADING Q MAY BE DELETED VIA:  V←1↓, ...
⍝         THEN THE EXAMPLE CAN BE SIMPLIFIED TO:
```

```
V←1↓,'/', 3 2 ⍴V
```

```
⍝ NOTE:  MAY NEED TO DELETE TRAILING OCCURRENCES OF Q AND/OR
⍝         EXTRANEOUS ELEMENTS OF V PRODUCED BY THE RESHAPE:
```

```
V←(¯1+R+⍴V)⍴(((R←⌈(⍴V)÷K),K)⍴V),Q
```

RESULT:

```
12/31/78
```

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

INSERTING A STRING S AFTER THE ELEMENT WITH INDEX I

```
V←((I+~⎕IO)↑V),S,(I+~⎕IO)↓V

          ⎕IO←1:   V←(I↑V),S,I↓V
          ⎕IO←0:   V←((I+1)↑V),S,(I+1)↓V
```

```
⍝ S MAY BE A SCALAR OR VECTOR
```

EXAMPLE:
```
V←'ABCDHIJ'
S←'EFG'
I←4         [⎕IO←1]
```

RESULT:
```
ABCDEFGHIJ
```

```
⍝ ALTERNATE IMPLEMENTATION:

      V←(V,S)[⍋(⍳⍴V),(⍴S)⍴I]          **

⍝ TO INSERT S AFTER SEVERAL ELEMENTS:

      N←(⍴S←,S)×⍴I←,I
      V←(V,N⍴S)[⍋(⍳⍴V),,⍉((⍴S),⍴I)⍴I]
```

EXAMPLE:
```
V←'B←(B=0)+B÷15'
S←'[3]'
I←3 8 0         [⎕IO←0]
```

RESULT:
```
B[3]←(B[3]=0)+B[3]÷15
```

```
⍝ NOTE:  SEE ALSO THE TECHNIQUE "REPLACING MULTIPLE
⍝    OCCURRENCES ..."
```

** ADAPTED FROM (1) THE APL IDIOM LIST

EXPANDING / INSERTING ELEMENTS INTO A VECTOR V

EXPANDING V TO LENGTH L

PADDING WITH THE PAD-ELEMENT (0 *OR BLANK*)

```
V←L↑V
```

EXAMPLE: V←⍳7
             L←15

RESULT: 1 2 3 4 5 6 7 0 0 0 0 0 0 0 0

⍝ *ALTERNATE IMPLEMENTATION*:

```
V←L↑V,L⍴Q        [Q←0 OR Q←' ']
```

PADDING WITH THE LAST ELEMENT

```
V←L↑V,L⍴¯1↑V
```

RESULT: 1 2 3 4 5 6 7 7 7 7 7 7 7 7 7

⍝ *ALTERNATE IMPLEMENTATION*:

```
V←V[(⍳L)⌊(⍴V)-~⎕IO]
```

TESTING THE ELEMENTS IN A VECTOR V

DETERMINING IF ELEMENT Q OCCURS IN V

```
R←Q∊V
```

```
⍝ RESULT R IS 1 IF Q OCCURS IN V AND 0 IF IT DOESN'T
⍝ IF Q IS A VECTOR, R IS A VECTOR OF THE SAME LENGTH
```

EXAMPLE:
```
V←'WE THE PEOPLE'
Q←'LATE'
```

RESULT: 1 0 1 1

```
⍝ ALTERNATE IMPLEMENTATIONS, WHEN Q IS A SCALAR:

R←∨/V=Q          OR      R←V∨.=Q

R←0<+/V=Q        OR      R←0<V+.=Q

⍝ ALTERNATE IMPLEMENTATIONS, WHEN Q IS EITHER VECTOR
⍝  OR SCALAR:

R←(⎕IO+⍴V)>V⍳Q

     ⎕IO←1:  R←(1+⍴V)>V⍳Q
     ⎕IO←0:  R←(⍴V)>V⍳Q

R←∨/Q∘.=V

R←((⍴V)-~⎕IO)≥V⍳Q

     ⎕IO←1:  R←(⍴V)≥V⍳Q
     ⎕IO←0:  R←((⍴V)-1)≥V⍳Q
```

TESTING THE ELEMENTS IN A VECTOR V

DETERMINING IN WHICH FIELD OF V AN ELEMENT Q OCCURS

```
⍝ FIND THE INDICES OF THE OCCURRENCES OF THE ELEMENT Q
⍝  IN V;  RANGE-CHECK THESE INDICES AGAINST THE INDICES
⍝  OF THE FIELD BOUNDARIES


      OC←≠/(I←(V=Q)/⍳⍴V)∘.≥FB


⍝ "FB" IS AN N×2 MATRIX WHERE EACH ROW CONTAINS THE LOWER
⍝  AND UPPER BOUNDARY INDICES OF A FIELD (SEE TECHNIQUE
⍝  "FINDING THE BOUNDARY INDICES ...").  THE INDICES USED
⍝  FOR THESE BOUNDS ARE DEPENDENT ON THE RELATIONAL
⍝  FUNCTION USED (> OR ≥).

⍝ NOTE:  "≠/N∘.≥R" IS A "KERNEL" APL OPERATION FOR
⍝   RANGE-CHECKING THE NUMBERS N AGAINST THE LOWER AND
⍝   UPPER RANGE LIMITS R
```

```
EXAMPLE:    V←'55.55 59.55 59.59 50.50 50.59 90.09'
            FB←6 2⍴1 6 7 12 13 18 19 24 25 30 31 36   [⎕IO←1]
            Q←'9'

RESULT:     BOUNDS:         6 12 18 24 30 36
                            1  7 13 19 25 31

            INDICES:   8    0  1  0  0  0  0
                      14    0  0  1  0  0  0
                      17    0  0  1  0  0  0
                      29    0  0  0  0  1  0
                      31    0  0  0  0  0  1
                      35    0  0  0  0  0  1
```

```
⍝ THE RESULTANT BIT MATRIX "OC" HAS 1 ROW FOR EACH INDEX
⍝  OCCURRENCE OF Q, AND 1 COLUMN FOR EACH FIELD DEFINED
⍝  IN "FB", I.E., ⍴OC ←→ (⍴I),1↑⍴FB.  A 0 OR 1 INDICATES
⍝  THE ABSENCE OR PRESENCE OF THAT OCCURRENCE OF Q IN THE
⍝  FIELD

⍝ EACH COLUMN OF "OC" CAN BE USED TO SELECT THE INDICES
⍝  THAT OCCUR IN FIELD "K", I.E.:   OC[;K]/I
```

(CONTD ON NEXT PAGE)

TESTING THE ELEMENTS IN A VECTOR V

DETERMINING IN WHICH FIELD OF V AN ELEMENT Q OCCURS (CONTD)

```
⍝ THE FIRST OCCURRENCE OF Q IN FIELD "K" CAN BE DETERMINED
⍝  USING:       T←<⍀OC
⍝  AND THEN:    T[;K]/I

⍝ ∨⌿OC  INDICATES WHICH FIELDS CONTAIN Q

⍝ THE COLUMNS CAN BE SUMMED TO COUNT THE NUMBER OF
⍝  OCCURRENCES OF Q IN THE FIELDS, I.E.:   +⌿OC


⍝ TO TEST FOR ANY OF SEVERAL DIFFERENT ELEMENTS "DQ",
⍝  REPLACE "V=Q"  WITH  "V∊DQ"  OR  "∨/V∘.=DQ"

⍝ THEN  ∨/OC  INDICATES WHICH ELEMENTS OF "DQ" OCCUR IN
⍝  SOME FIELD
```

TESTING THE ELEMENTS IN A VECTOR V

COUNTING THE NUMBER OF OCCURRENCES N OF ELEMENTS Q IN V

```
N←+/Q∘.=V
```

```
⍝ "N" CONTAINS THE COUNTS FOR EACH RESPECTIVE ELEMENT IN Q
⍝ IF Q IS A SINGLE ELEMENT, THIS SIMPLIFIES TO:
```

```
N←+/Q=V          OR      N←Q+.=V
```

EXAMPLE:

```
V←'ALPHA,BETA,GAMMA,DELTA,EPSILON'
Q←',AFIT'
```

RESULT:

```
4 6 0 1 2
```

```
⍝ TO COUNT THE GRAND TOTAL OF ALL OCCURRENCES OF ELEMENTS Q
⍝  IN V:
```

```
N←+/V∊Q
```

RESULT:

```
13
```

TESTING THE ELEMENTS IN A VECTOR V

PERFORMING RELATIONAL TESTS ALONG V

```
R←(¯1↓V) α 1↓V

⍝ THIS REPRESENTS:  (V[1]αV[2]),(V[2]αV[3]),...,
⍝                          V[¯1+ρV]αV[ρV]       [⎕IO←1]

⍝ "α" IS ONE OF THE RELATIONAL FUNCTIONS  < ≤ = ≠ ≥ >
⍝   FOR NUMBERS AND  = ≠  FOR CHARACTERS

⍝ TO TEST IF THE RELATION HOLDS ALONG V, SUBSTITUTE
⍝   R←∧/...   OR   ... ∧.α ...

⍝ ρR ←→ (ρV)-1

⍝ REDUCTION (α/V) DOES NOT PRODUCE THIS RESULT, SINCE ITS
⍝  INTERMEDIATE RIGHT-COMPARANDS BECOME BOOLEAN AFTER THE
⍝  RIGHTMOST COMPARISON
```

EXAMPLE: TESTING FOR "<" ALONG V

```
V←1 2 4 6 5 9 13 17
```

RESULT: 1 1 1 0 1 1 1

```
⍝ ALTERNATE IMPLEMENTATIONS [ρR ←→ ρV]:

⍝ FOR ≤:     R←(⍋V)=⍳ρV          **

⍝ FOR ALL =:    R←∧/V=1ρV      OR      R←V∧.=1ρV          **
```

** ADAPTED FROM (1) *THE APL IDIOM LIST*

TESTING THE ELEMENTS IN A VECTOR V

DETERMINING IF ALL THE ELEMENTS OF V ARE UNIQUE

```
R←∧/(⍳⍴V)=V⍳V          **
```

EXAMPLE: `V←'⊂⊃∩∪⊥⊤|⍺⌈⌊∇∆∘⎕*○⍳↓↑~⍴∊⍵⍺?<≤=≥>≠∨∧'`

RESULT: 0

⍝ *ALTERNATE IMPLEMENTATION*:

```
R←~0∊(⍳⍴V)=V⍳V
```

** *ADAPTED FROM* (1) *THE APL IDIOM LIST*

SELECTING ELEMENTS OF A VECTOR V

SELECTING ELEMENTS THAT SATISFY A TEST

```
R←( TEST )/V
```

```
⍝ THE "TEST" MUST PRODUCE A BOOLEAN RESULT (E.G., USING
⍝  A LOGICAL OR RELATIONAL EXPRESSION) WHICH IS A SCALAR
⍝  OR A VECTOR WITH THE SAME LENGTH AS V

⍝ IF NO ELEMENTS ARE SELECTED, THE RESULT IS AN EMPTY VECTOR
```

EXAMPLE: SELECTING ELEMENTS > 5

```
R←(V>5)/V←2 5 8 13 19
```

RESULT: 8 13 19

EXAMPLE: SELECTING ELEMENTS ≠ BLANKS

```
R←(V≠' ')/V←' THIS IS A CHAR. LINE'
```

RESULT: THISISACHAR.LINE

EXAMPLE: SELECTING / DELETING A SUBSTRING

```
R←'**DATE',(( TEST )/'-RANGE'),' ERROR'
TEST←0
```

RESULT: **DATE ERROR

```
⍝ NOTE:  THE CODE FOR OTHER "TESTS" MAY BE EXTRACTED FROM
⍝   THE APPROPRIATE TECHNIQUES IN THIS GUIDE.  FOR
⍝   EXAMPLE, THE TEST FOR "ALL LEADING OCCURRENCES" MAY BE
⍝   FOUND IN THE TECHNIQUE FOR DELETING SUCH OCCURRENCES.

⍝   ADDITIONAL TESTS ARE SHOWN ON THE FOLLOWING PAGES.
```

SELECTING ELEMENTS OF A VECTOR V

SELECTING ELEMENTS P WHICH ARE BETWEEN PAIRED DELIMITERS D

TYPE 1: ONLY 1 TYPE OF DELIMITER ELEMENT

```
P←(B≥1↓≠\1,B←D=V)/V
```

```
⍝ THE RESULT WILL INCLUDE THE DELIMITERS AND THE ELEMENTS
⍝  BETWEEN THEM

⍝ CAUTION:  NO "NESTING" OF PAIRED DELIMITERS IS ALLOWED
```

EXAMPLE: BETWEEN PAIRED QUOTATION MARKS (")

```
V←'THE "BEST" ANSWER IS "LIMIT".'
D←'"'
```

RESULT: "BEST""LIMIT"

```
⍝ ALTERNATE IMPLEMENTATION:

P←(C≤1↓=\0,C←D≠V)/V

⍝ TO SELECT WITHOUT THE DELIMITERS:

P←(B⍱1↓≠\1,B←D=V)/V

P←(C∧1↓=\0,C←D≠V)/V
```

RESULT: BESTLIMIT

```
⍝ NOTE:  THE TEST FOR BALANCED DELIMITERS IS:

0=2|+/B        OR        0=2|+/D=V
```

(CONTD ON NEXT PAGE)

SELECTING ELEMENTS OF A VECTOR V

SELECTING ELEMENTS P WHICH ARE BETWEEN PAIRED DELIMITERS D (CONTD)

TYPE 2: LEFT AND RIGHT DELIMITER ELEMENTS

```
P←(L≤+\(V=1↑D)-¯1↓0,V=1↓D)/V
```

```
⍝ "D" CONTAINS THE DELIMITERS IN THE ORDER OF PAIRING,
⍝   E.G., D←'[]'
⍝ THE RESULT WILL INCLUDE THE DELIMITERS AND THE ELEMENTS
⍝   BETWEEN THEM
⍝ "L" IS THE LEVEL OF NESTING OF THE PAIRED DELIMITERS
⍝   TO BE SELECTED
```

EXAMPLE: BETWEEN PARENTHESES

```
V←'((⍳⍴R)=R⍳R)/R←(5|N)*2'
D←'()'
L←1
```

RESULT: `((⍳⍴R)=R⍳R)(5|N)`

```
⍝ ALTERNATE IMPLEMENTATION:

P←(L≤+\1↓-⌿ 0 ¯1 ⌽0,D∘.=V)/V

⍝ TO SELECT WITHOUT THE DELIMITERS FOR LEVEL "L":

P←(L≤+\(¯1↓0,V=1↑D)-V=1↓D)/V

P←(L≤+\1↓-⌿ ¯1 0 ⌽0,D∘.=V)/V
```

RESULT: `(⍳⍴R)=R⍳R5|N`

```
⍝ NOTE: TESTS FOR BALANCED DELIMITERS ARE:

∧/0≤+\(V=1↑D)-V=1↓D       OR       ∧/0≤+\-⌿D∘.=V
```

SELECTING ELEMENTS OF A VECTOR V

SELECTING ELEMENTS P WHICH ARE OUTSIDE PAIRED DELIMITERS D

TYPE 1: *ONLY 1 TYPE OF DELIMITER ELEMENT*

```
P←(B∨1↓≠\1,B←D=V)/V
```

```
⍝ THE RESULT WILL INCLUDE THE DELIMITERS AND THE ELEMENTS
⍝  OUTSIDE THEM

⍝ CAUTION:  NO "NESTING" OF PAIRED DELIMITERS IS ALLOWED
```

EXAMPLE: OUTSIDE PAIRED QUOTATION MARKS (")

```
V←'THE "BEST" ANSWER IS "LIMIT".'
D←'"'
```

RESULT: THE "" ANSWER IS "".

```
⍝ ALTERNATE IMPLEMENTATION:

P←(C⍲1↓=\0,C←D≠V)/V

⍝ TO SELECT WITHOUT THE DELIMITERS:

P←(B<1↓≠\1,B←D=V)/V

P←(C>1↓=\0,C←D≠V)/V
```

RESULT: THE ANSWER IS .

(CONTD ON NEXT PAGE)

SELECTING ELEMENTS OF A VECTOR V

SELECTING ELEMENTS P WHICH ARE OUTSIDE PAIRED DELIMITERS D (*CONTD*)

TYPE 2: *LEFT AND RIGHT DELIMITER ELEMENTS*

```
P←(L>+\(¯1↓0,V=1↑D)-V=1↓D)/V
```

```
⍝ "D" CONTAINS THE DELIMITERS IN THE ORDER OF PAIRING,
⍝   E.G., D←'[]'
⍝ THE RESULT WILL INCLUDE THE DELIMITERS AND THE ELEMENTS
⍝   OUTSIDE THEM
⍝ "L" IS THE LEVEL OF NESTING OF THE PAIRED DELIMITERS
⍝   TO BE SELECTED
```

EXAMPLE: *OUTSIDE PARENTHESES*

```
V←'((⍳⍴R)=R⍳R)/R←(5|N)*2'
D←'()'
L←1
```

RESULT:

```
()/R←()*2
```

```
⍝ ALTERNATE IMPLEMENTATION:
```

```
P←(L>+\1↓-⌿ ¯1 0 ⌽0,D∘.=V)/V
```

```
⍝ TO SELECT WITHOUT THE DELIMITERS FOR LEVEL "L":
```

```
P←(L>+\(V=1↑D)-¯1↓0,V=1↓D)/V
```

```
P←(L>+\1↓-⌿ 0 ¯1 ⌽0,D∘.=V)/V
```

RESULT:

```
/R←*2
```

SELECTING ELEMENTS OF A VECTOR V

SELECTING SUBSTRING S WITH STARTING INDEX I AND LENGTH L

```
        S←L↑(I-⎕IO)↓V
```

```
EXAMPLE:    V←'THESE ARE THE TIMES'
            I←6       [⎕IO←1]
            L←8

RESULT:      ARE THE
```

```
⍝ ALTERNATE IMPLEMENTATION:

        S←V[(I-⎕IO)+⍳L]

⍝ TO SELECT MULTIPLE SUBSTRINGS WITH SAME LENGTH:

        S←V[,(I-⎕IO)∘.+⍳L]
```

```
EXAMPLE:    I←8 1 17      [⎕IO←1]
            L←3

RESULT:     RE THEMES
```

SELECTING ELEMENTS OF A VECTOR V

SELECTING LEFT AND RIGHT FIELDS OF V DELIMITED BY THE ELEMENT Q

```
R←(1+ρL←((V⍳Q)-⎕IO)↑V)↓V
```

EXAMPLE:
```
V←'PARM1,ARG2'
Q←','
```

RESULT:
```
L: PARM1
R: ARG2
```

SELECTING ELEMENTS OF A VECTOR V

SELECTING FIELD N OF V

TYPE 1: *USING A FIELD PARTITION VECTOR P*

```
⍝ MUST FIRST GENERATE A LOGICAL VECTOR P DEFINING THE
⍝  FIELD PARTITION OF V, WITH A 1-BIT FOR THE 1ST
⍝  ELEMENT OF EACH FIELD OF V AND 0'S OTHERWISE
⍝  (SEE SECTION "GENERATING A LOGICAL VECTOR").
⍝ ALSO  ⍴P ←→ ⍴V

⍝ CAUTION:  ANY FIELD DELIMITERS MUST BE DELETED FROM V
```

```
      F←(N=+\P)/V
```

```
⍝ "N" IS THE NUMBER OF THE FIELD

⍝ NOTE:  "+\P" IS A "KERNEL" APL OPERATION
```

EXAMPLE:
```
V←'1111122233444556666667'
P←1 0 0 0 0 1 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1
N←6
```

RESULT: 666666

```
⍝ ALTERNATE IMPLEMENTATIONS, WHEN N IS A VECTOR:
```

```
      F←((+\P)∊N)/V

      F←(∨⌿N∘.=+\P)/V
```

EXAMPLE: N←2 5 7

RESULT: 222557

(*CONTD ON NEXT PAGE*)

SELECTING ELEMENTS OF A VECTOR V

SELECTING FIELD N OF V (*CONTD*)

TYPE 2: *USING THE INDEX OF THE FIRST ELEMENT AND THE FIELD WIDTH*

```
⍝ PERFORM THE FOLLOWING 3 TECHNIQUES:
⍝       1. COMPUTE WIDTHS W OF THE FIELDS OF V
⍝       2. FIND THE STARTING (BOUNDARY) INDICES I OF EACH
⍝          FIELD
⍝       3. USE THESE TO SELECT THE SUBSTRING "FIELD N" VIA:

              F←W[N]↑(I[N]-⎕IO)↓V

⍝ "N" IS THE INDEX-NUMBER OF THE FIELD
```

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE INDICES OF ELEMENTS THAT SATISFY A TEST

```
I←( TEST )/⍳⍴V
```

```
⍝ NOTE:  THIS IS A "KERNEL" APL OPERATION
```

EXAMPLE: FINDING THE INDICES OF THE DELIMITER "∩"

```
V←'∩ALPHA∩BETA∩GAMMA∩PI∩ZETA'
I←(V='∩')/⍳⍴V
```

RESULT: 1 7 12 18 21 [⎕IO←1]

```
⍝ NOTE:  TO OBTAIN THE INDICES FOR A SPECIFIC
⍝    INDEX ORIGIN, ADD:
```

```
⎕IO←1:  I←(~⎕IO)+( TEST )/⍳⍴V

⎕IO←0:  I←(-⎕IO)+( TEST )/⍳⍴V
```

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE INDEX OF THE 1ST OCCURRENCE OF ELEMENTS Q

```
I←V⍳Q
```

EXAMPLE:

```
V←'EDCBAJIHGFEJDIZG'
Q←'GADH'
```

RESULT: 9 5 2 8 [⎕IO←1]

```
⍝ ALTERNATE IMPLEMENTATION:

      I←⎕IO++/∧\Q∘.≠V

⍝ ALTERNATE IMPLEMENTATIONS, WHEN Q IS A SCALAR OR
⍝  1-ELEMENT VECTOR:

      I←1↑(V=Q)/⍳⍴V

      I←(<\V=Q)/⍳⍴V
```

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE INDEX OF THE LAST OCCURRENCE OF ELEMENTS Q

```
I←(¯1 1[2×⎕IO]+⍴V)-(⌽V)⍳Q

     ⎕IO←1:   I←(1+⍴V)-(⌽V)⍳Q
     ⎕IO←0:   I←(¯1+⍴V)-(⌽V)⍳Q
```

EXAMPLE:

```
V←'EDCBAJIHGFEJDIZG'
Q←'GADHX'
```

RESULT:

```
16 5 13 8 0     [⎕IO←1]
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

      I←(((2×⎕IO)-1)+⍴V)-(⌽V)⍳Q

      I←((¯1*~⎕IO)+⍴V)-(⌽V)⍳Q

      I←(+/∨\Q∘.=⌽V)-~⎕IO

⍝ ALTERNATE IMPLEMENTATION, WHEN Q IS A SCALAR OR
⍝  1-ELEMENT VECTOR:

      I←¯1↑(V=Q)/⍳⍴V
```

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE INDICES I OF THE SMALLEST AND LARGEST ELEMENTS

⍝ *THE SMALLEST ELEMENT*:

```
I←V⍳⌊/V          **
```

⍝ *NOTE*: *FINDS INDEX OF THE 1ST OCCURRENCE*

EXAMPLE: V←14 76 46 54 22 5 68 68 94 39

RESULT: 6 [⎕IO←1]

⍝ *ALTERNATE IMPLEMENTATIONS*:

```
I←1↑⍋V    OR    I←¯1↑⍒V
```

⍝ *NOTE*: "...¯1↑..." *FINDS THE LAST OCCURRENCE*

⍝ *THE LARGEST ELEMENT*:

```
I←V⍳⌈/V          **
```

RESULT: 9 [⎕IO←1]

⍝ *ALTERNATE IMPLEMENTATIONS*:

```
I←1↑⍒V    OR    I←¯1↑⍋V
```

** *ADAPTED FROM* (1) *THE APL IDIOM LIST*

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE STARTING INDICES I OF OCCURRENCES OF A PATTERN OF ELEMENTS

FINDING ALL OCCURRENCES OF A SUBSTRING S

```
⍝ FIND INDICES OF ALL OCCURRENCES OF 1ST ELEMENT OF
⍝  SUBSTRING, EXCEPT ANY THAT ARE TOO CLOSE TO END OF V

        I←(L↓V=1↑S)/⍳(⍴V)+L←1-⍴S←,S

⍝ EXTRACT THE SUBSTRINGS AND KEEP ONLY THE INDICES OF
⍝  THE IDENTICAL ONES

        I←(V[(I-⎕IO)∘.+⍳⍴S]∧.=S)/I

⍝ NOTE:  IF "⍴I" WILL BE GREATER THAN "⍴S", CODE
⍝     "...V[I∘.+(⍳⍴S)-⎕IO]..."
```

EXAMPLE: V←'STORE THE FIRST INSTANCE OF "ST" LAST'
S←'ST'

RESULT: 1 14 19 30 36 [⎕IO←1]

```
⍝ RESULT IS THE EMPTY VECTOR WHEN THE SUBSTRING IS NOT
⍝  FOUND

⍝ ALTERNATE IMPLEMENTATIONS:

        I←((-L)↓S∧.=(L,1+⍴V)⍴V)/⍳(⍴V)+1-L←⍴S←,S          **

        I←(∧⌿((⍳⍴S)-⎕IO)⌽(S←,S)∘.=V)/⍳⍴V

⍝ NOTE:  THIS LAST IMPLEMENTATION FINDS OCCURRENCES
⍝    "WRAPPED AROUND" THE ENDS OF V.  TO AVOID THIS:

        I←(1↓∧⌿((⍳⍴S)-⎕IO)⌽0,(S←,S)∘.=V)/⍳⍴V
```

** ADAPTED FROM (3) *USE AND MISUSE OF APL*

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE STARTING INDICES I OF OCCURRENCES OF A PATTERN OF ELEMENTS

FINDING ALL OCCURRENCES OF A WORD S

```
⍝ NOTE:  IGNORES ANY OCCURRENCE OF THE WORD THAT IS
⍝   EMBEDDED WITHIN ANOTHER WORD


⍝ FIND INDICES OF ALL OCCURRENCES OF 1ST ELEMENT OF
⍝  WORD, EXCEPT ANY THAT ARE TOO CLOSE TO END OF V

      I←(L↓V=1↑S)/⍳(⍴V)+L←1-⍴S←,S

⍝ EXTRACT THE SUBSTRINGS AND KEEP ONLY THE INDICES
⍝  OF THE IDENTICAL ONES

      I←(V[(I-⎕IO)∘.+⍳⍴S]∧.=S)/I

⍝ COMPUTE THE INDICES OF THE ELEMENTS IMMEDIATELY
⍝  BEFORE AND AFTER EACH SUBSTRING, AND RANGE-CHECK

      L←,≠/(J←I∘.+¯1,⍴S)∘.>(0,⍴V)-~⎕IO

⍝ KEEP ONLY THE STARTING INDICES OF THE SUBSTRINGS
⍝  WHOSE ADJACENT ELEMENTS ARE NON-WORD ELEMENTS OR
⍝  AT THE VECTOR ENDS

⍝  NOTE: "ALF" IS A VECTOR OF VALID ELEMENTS FOR
⍝    WORDS OR LABELS

      I←(~∨/(⍴J)⍴L\V[L/,J]∊ALF)/I
```

EXAMPLE:

```
V←'STORE THE FIRST INSTANCE OF "ST" LAST'
S←'ST'
ALF←'ABCDEFGHIJKLMNOPQRSTUVWXYZ∆0123456789'
ALF←ALF,'A̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲⍙'
```

RESULT: 30   [⎕IO←1]

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE BOUNDARY INDICES OF THE FIELDS OF V

WHEN THERE ARE NO FIELD DELIMITERS

```
TYPE 1:  FROM A VECTOR OF FIELD-WIDTHS W, WHERE ⍴V ←→ +/W

FIRST ELEMENTS:  LB←+\¯1↓⎕IO,W

LAST ELEMENTS:   UB←(+\W)-~⎕IO

EXAMPLE:    W←5 2 3 1 4

RESULT:     LB:  1 6 8 11 12      [⎕IO←1]
            UB:  5 7 10 11 15

            LB:  0 5 7 10 11      [⎕IO←0]
            UB:  4 6 9 10 14

⍝ NOTE:  FOR ¨K¨ FIELDS WITH SAME WIDTH ¨L¨, SET ¨W←K⍴L¨

⍝ NOTE:  TO OBTAIN THE INDICES FOR A SPECIFIC
⍝   INDEX ORIGIN, ADD:

               ⎕IO←1:  LB←(~⎕IO)+...

               ⎕IO←0:  LB←(-⎕IO)+...

⍝   [LIKEWISE FOR ¨UB¨]
```

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE BOUNDARY INDICES OF THE FIELDS OF V (CONTD)

WHEN THERE ARE NO FIELD DELIMITERS (CONTD)

TYPE 2: WHEN V CONTAINS FIELDS OF IDENTICAL ELEMENTS

```
FIRST ELEMENTS:  LB←(1,(1↓V)≠¯1↓V)/⍳⍴V

LAST ELEMENTS:   UB←(((1↓V)≠¯1↓V),1)/⍳⍴V

⍝ NOTE:  IF "⍳⍴V" IS REPLACED BY "V", THE BOUNDARY
⍝    ELEMENTS (FIRST OR LAST) OF EACH FIELD WILL BE
⍝    SELECTED
```

EXAMPLE: `V←'AABBBBCDDDEEEE'`

```
RESULT:     LB:  1 3 7 8 11      [⎕IO←1]
            UB:  2 6 7 10 14

⍝ NOTE:  TO OBTAIN THE INDICES FOR A SPECIFIC
⍝    INDEX ORIGIN, ADD:

               ⎕IO←1:  LB←(~⎕IO)+...

               ⎕IO←0:  LB←(-⎕IO)+...

⍝ ALTERNATE IMPLEMENTATIONS:

FIRST:   LB←(V≠¯1↓(1↑0⍴V),V)/⍳⍴V

LAST:    UB←(V≠1↓V,1↑0⍴V)/⍳⍴V

⍝ THESE CAN BE SIMPLIFIED AS FOLLOWS:

         NUMERIC V                  CHARACTER V

FIRST:   LB←(V≠¯1↓0,V)/⍳⍴V          LB←(V≠¯1↓' ',V)/⍳⍴V

LAST:    UB←(V≠1↓V,0)/⍳⍴V           UB←(V≠1↓V,' ')/⍳⍴V
```

(CONTD ON NEXT PAGE)

FINDING INDICES OF ELEMENTS OF A VECTOR V

FINDING THE BOUNDARY INDICES OF THE FIELDS OF V (CONTD)

WHEN THERE ARE NO FIELD DELIMITERS (CONTD)

TYPE 2: *WHEN V CONTAINS FIELDS OF IDENTICAL ELEMENTS (CONTD)*

```
⍝ CAUTION:  1ST AND LAST ELEMENTS OF V MUST BE
⍝   DIFFERENT TO USE THE FOLLOWING IMPLEMENTATIONS
```

```
FIRST:  LB←(V≠¯1⌽V)/⍳⍴V

LAST:   UB←(V≠1⌽V)/⍳⍴V
```

WHEN THE FIELD DELIMITER IS Q

```
FIRST ELEMENTS:  LB←(1,V=Q)/⍳1+⍴V

LAST ELEMENTS:   UB←¯1+((V=Q),1)/⍳1+⍴V
```

```
⍝ ASSUMES THAT V CONTAINS NO LEADING OR TRAILING
⍝  DELIMITERS Q, UNLESS IT HAS EMPTY FIELDS.

⍝ TO PRODUCE AN N×2 MATRIX ¨FB¨ WHERE EACH ROW
⍝  CONTAINS THE BOUNDARY INDICES OF A FIELD,
⍝  ENTER:  FB←LB,[⎕IO+0.5]UB
```

```
EXAMPLE:    V←'55.55 59.55 59.59 50.50 50.59 90.09'
            Q←' '

RESULT:     LB: 1 7 13 19 25 31      [⎕IO←1]
            UB: 5 11 17 23 29 35
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

FIRST:  LB←(U=Q)/⍳⍴U←Q,V

LAST:   UB←¯1+(U=Q)/⍳⍴U←V,Q
```

SHIFTING A VECTOR V

SHIFTING N POSITIONS WITHOUT CHANGING THE LENGTH OF V

WITH WRAP-AROUND (ROTATING)

```
RIGHT:    V←(-N)⌽V

LEFT:     V←N⌽V

EXAMPLE:    V←⍳10
            N←3

RESULT:     R:  8 9 10 1 2 3 4 5 6 7
            L:  4 5 6 7 8 9 10 1 2 3

⍝ NOTE:  USING ONLY    V←N⌽V
⍝    LEFT ↔ N>0  AND  RIGHT ↔ N<0
```

WITHOUT WRAP-AROUND

```
⍝ PAD-ELEMENT WILL BE 0 FOR NUMBERS, BLANKS FOR CHARS

RIGHT:    V←(-⍴V)↑(-N)↓V

LEFT:     V←(⍴V)↑N↓V

EXAMPLE:    V←⍳10
            N←4

RESULT:     R:  0 0 0 0 1 2 3 4 5 6
            L:  5 6 7 8 9 10 0 0 0 0
```

SHIFTING A VECTOR V

CENTERING V

WHEN THE FILL-ELEMENT IS Q

```
V←(⌈0.5×(B⍳1)-(⌽B←V≠Q)⍳1)⌽V
```

EXAMPLE: V←'--AB-C---D--E-----'
Q←'-'

RESULT: ---AB-C---D--E----

SHIFTING A VECTOR V

CENTERING V

WITHIN A FIELD OF PAD-ELEMENTS OF WIDTH W

```
W↑(-⌊(W+⍴V)÷2)↑V
```

⍝ *OMIT THE "W↑" IF DO NOT NEED RESULT IN FULL WIDTH W*

<u>EXAMPLE</u>:
```
V←'TABLE   OF   CONTENTS'
W←30
```

<u>RESULT</u>:
```
|      TABLE   OF   CONTENTS      |
```

⍝ *ALTERNATE IMPLEMENTATIONS:*

```
W↑((⌈0.5×W-⍴V)-W)↑V

W↑((⌊0.5×W-⍴V)⍴Q),V      [Q←0 OR Q←' ']
```

SHIFTING A VECTOR V

CENTERING V

COMPRESSING THE FILL-ELEMENT Q TO THE SIDES

```
V←(⌈0.5×+/~B)⌽V[⍋B←V≠Q]
```

EXAMPLE:

```
V←'∘∘A∘∘∘BC∘D∘∘EF∘∘∘∘∘G'
Q←'∘'
```

RESULT:

```
∘∘∘∘∘∘ABCDEFG∘∘∘∘∘∘∘
```

```
⍝ ALTERNATE IMPLEMENTATION:

V←(⌈0.5×+/~B)⌽(-⍴V)↑((⍴V)⍴Q),(B←V≠Q)/V
```

SHIFTING A VECTOR V

RIGHT-JUSTIFYING V

WHEN THE FILL-ELEMENT IS Q

```
V←(⎕IO-(Q≠⌽V)⍳1)⌽V

        ⎕IO←1:  V←(1-(Q≠⌽V)⍳1)⌽V
        ⎕IO←0:  V←(-(Q≠⌽V)⍳1)⌽V
```

```
⍝ THE COMPLEMENT IS:  V←(⎕IO-(Q=⌽V)⍳0)⌽V
```

EXAMPLE:
```
V←'***A*BB**CCC*****'
Q←'*'
```

RESULT:
```
********A*BB**CCC
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

V←((¯1↑(V≠Q)/⍳⍴V)+~⎕IO)⌽V

V←(1-(V=Q)⊥1)⌽V          **

V←(-+/∧\Q=⌽V)⌽V
```

```
⍝ NOTE:  IF ¨Q¨ IS A VECTOR OF DIFFERENT FILL-ELEMENTS,
⍝   REPLACE ¨V≠Q¨ WITH ¨~V∊Q¨ AND ¨Q≠⌽V¨ WITH ¨~(⌽V)∊Q¨
```

** ADAPTED FROM (1) *THE APL IDIOM LIST*

(CONTD ON NEXT PAGE)

SHIFTING A VECTOR V

RIGHT-JUSTIFYING V

WHEN THE FILL-ELEMENT IS Q (CONTD)

```
⍝ SPECIAL CASE IMPLEMENTATION:

⍝   WHEN V IS LEFT-JUSTIFIED AND THERE ARE NO
⍝    EMBEDDED FILL-ELEMENTS Q

        V←(+/V≠Q)⌽V     OR     V←(V+.≠Q)⌽V
```

EXAMPLE:
```
V←'ALASKA*******'
Q←'*'
```

RESULT:
```
*******ALASKA
```

```
⍝ ALTERNATE IMPLEMENTATION:

        V←((V⍳Q)-⎕IO)⌽V
```

SHIFTING A VECTOR V

RIGHT-JUSTIFYING V

WITHIN A FIELD OF PAD-ELEMENTS OF WIDTH W

```
(-W)↑V
```

```
⍝ NOTE:  IF W < ⍴V, V WILL BE TRUNCATED ON THE LEFT

EXAMPLE:    V←⍳7
            W←12

RESULT:     0 0 0 0 0 1 2 3 4 5 6 7

⍝ NOTE:  THE FORMAT FUNCTION "⍕" CAN ALSO BE USED TO
⍝    RIGHT-JUSTIFY THE CHARACTER RESULT WHEN FORMATTING
⍝    NUMBERS

⍝ ALTERNATE IMPLEMENTATION:

        (-W)↑(W⍴Q),V      [Q←0 OR Q←' ']
```

SHIFTING A VECTOR V

RIGHT-JUSTIFYING V

COMPRESSING THE FILL-ELEMENT Q LEFT

```
V←V[⍋V≠Q]
```

EXAMPLE:

```
V←'***A*BB**CCC*****'
Q←'*'
```

RESULT:

```
***********ABBCCC
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

V←(B/V),(~B←V=Q)/V

V←((+/~B)⍴Q),(B←V≠Q)/V

V←(-⍴V)↑((⍴V)⍴Q),(V≠Q)/V

⍝ ALTERNATE IMPLEMENTATION, WHEN Q IS THE PAD-ELEMENT:

V←(-⍴V)↑(V≠Q)/V
```

EXAMPLE:

```
V←0 0 1 0 2 3 0 0 4 0
Q←0
```

RESULT:

```
0 0 0 0 0 0 1 2 3 4
```

SHIFTING A VECTOR V

LEFT-JUSTIFYING V

WHEN THE FILL-ELEMENT IS Q

```
V←(((V≠Q)⍳1)-⎕IO)⌽V

        ⎕IO←1:  V←(((V≠Q)⍳1)-1)⌽V
        ⎕IO←0:  V←((V≠Q)⍳1)⌽V

⍝ THE COMPLEMENT IS:  V←(((V=Q)⍳0)-⎕IO)⌽V
```

EXAMPLE: V←'***A*BB**CCC*****'
Q←'*'

RESULT: A*BB**CCC********

⍝ *ALTERNATE IMPLEMENTATIONS*:

```
V←((1↑(V≠Q)/⍳⍴V)-⎕IO)⌽V

V←(+/∧\V=Q)⌽V

V←(((Q=⌽V)⊥1)-1)⌽V
```

⍝ *NOTE*: IF "Q" IS A VECTOR OF DIFFERENT FILL-ELEMENTS,
⍝ REPLACE "V≠Q" WITH "~V∊Q" AND "Q≠⌽V" WITH "~(⌽V)∊Q"

(CONTD ON NEXT PAGE)

SHIFTING A VECTOR V

LEFT-JUSTIFYING V

WHEN THE FILL-ELEMENT IS Q (CONTD)

```
⍝ SPECIAL CASE IMPLEMENTATION:

⍝   WHEN V IS RIGHT-JUSTIFIED AND THERE ARE NO
⍝    EMBEDDED FILL-ELEMENTS Q

        V←(-+/V≠Q)⌽V      OR      V←(-V+.≠Q)⌽V
```

EXAMPLE: `V←'*******ALASKA'`
`Q←'*'`

RESULT: `ALASKA*******`

SHIFTING A VECTOR V

LEFT-JUSTIFYING V

WITHIN A FIELD OF PAD-ELEMENTS OF WIDTH W

```
W↑V
```

⍝ *NOTE*: *IF W < ρV, V WILL BE TRUNCATED ON THE RIGHT*

EXAMPLE:

```
V←⍳7
W←12
```

RESULT:

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

⍝ *ALTERNATE IMPLEMENTATION*:

```
W↑V,WρQ        [Q←0 OR Q←' ']
```

SHIFTING A VECTOR V

LEFT-JUSTIFYING V

COMPRESSING THE FILL-ELEMENT Q RIGHT

```
V←V[⍋V≠Q]
```

EXAMPLE:

```
V←'***A*BB**CCC*****'
Q←'*'
```

RESULT:

```
ABBCCC***********
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

      V←(B/V),(~B←V≠Q)/V

      V←(B/V),(+/~B←V≠Q)⍴Q

      V←(⍴V)↑((V≠Q)/V),(⍴V)⍴Q

⍝ ALTERNATE IMPLEMENTATION, WHEN Q IS THE PAD-ELEMENT:

      V←(⍴V)↑(V≠Q)/V
```

EXAMPLE:

```
V←0 0 1 0 2 3 0 0 4 0
Q←0
```

RESULT:

```
1 2 3 4 0 0 0 0 0 0
```

SHIFTING A VECTOR V

REVERSING THE POSITIONS OF THE ELEMENTS OF V

REVERSING THE ELEMENTS WITH INDICES I

```
V[⌽I]←V[I]
```

EXAMPLE:
```
V←13 56 35 44 21 60 78
I←2 5       [⎕IO←1]
```

RESULT:
```
13 21 35 44 56 60 78
```

```
⍝ NOTE:  IF 2<⍴I, THIS WILL REVERSE THE POSITIONS OF
⍝   THE GROUP OF ELEMENTS
```

SHIFTING A VECTOR V

REVERSING THE POSITIONS OF THE ELEMENTS OF V

REVERSING THE ELEMENTS WITHIN EACH FIELD OF V

```
⍝ MUST FIRST GENERATE A LOGICAL VECTOR P DEFINING THE
⍝  FIELD PARTITION OF V, WITH A 1-BIT FOR THE 1ST
⍝  ELEMENT OF EACH FIELD OF V AND 0'S OTHERWISE
⍝  (SEE SECTION "GENERATING A LOGICAL VECTOR").
⍝ ALSO ⍴P ←→ ⍴V

⍝ CAUTION:  ANY FIELD DELIMITERS MUST BE DELETED FROM V


      V←V[⌽⍒+\P]          **
```

EXAMPLE: V←'ABCDEFGHIJ'
P←1 0 0 1 1 0 0 0 1 0

RESULT: CBADHGFEJI

** ADAPTED FROM (2) *BOOLEAN FUNCTIONS AND TECHNIQUES*

SORTING A VECTOR V

SORTING A NUMERIC VECTOR

IN ASCENDING ORDER

```
V←V[⍋V]
```

```
⍝ CAUTION:  BEFORE SORTING, MAY HAVE TO USE ¨V←,V¨
⍝   TO ENSURE THAT V IS A VECTOR
```

EXAMPLE: V←77 29 42 3 18 81

RESULT: 3 18 29 42 77 81

```
⍝ NOTE:  CAN USE ¨I←⍋V¨ TO SORT ANOTHER VECTOR ¨W¨ WHOSE
⍝   ELEMENTS CORRESPOND TO V, I.E.:  W←W[I]
```

IN DESCENDING ORDER

```
V←V[⍒V]
```

RESULT: 81 77 42 29 18 3

```
⍝ NOTE:  IN SUBSEQUENT TECHNIQUES, A DESCENDING SORT
⍝  MAY BE OBTAINED BY SUBSTITUTING ¨⍒¨ FOR ¨⍋¨
```

IN EITHER ASCENDING OR DESCENDING ORDER

```
V←V[⍋V×T]        **
```

```
⍝ FOR ASCENDING SORT, SET ¨T←1¨; FOR DESCENDING, ¨T←¯1¨
```

EXAMPLE: T←¯1

RESULT: 81 77 42 29 18 3

** ADAPTED FROM (1) *THE APL IDIOM LIST*

SORTING A VECTOR V

SORTING A NUMERIC VECTOR

SORTING THE ELEMENTS OF V WITHIN THE FIELDS OF V

```
⍝ MUST FIRST GENERATE A LOGICAL VECTOR P DEFINING THE
⍝  FIELD PARTITION OF V, WITH A 1-BIT FOR THE 1ST
⍝  ELEMENT OF EACH FIELD OF V AND 0'S OTHERWISE
⍝  (SEE SECTION ¨GENERATING A LOGICAL VECTOR¨).
⍝ ALSO ⍴P ←→ ⍴V

⍝ CAUTION:  ANY FIELD DELIMITERS MUST BE DELETED FROM V
```

IN ASCENDING ORDER

```
        V←V[S[⍋(+\P)[S←⍋V]]]                        **

  EXAMPLE:    V←10 9 8 7 6 5 4 3 2 1
              P←1 0 0 1 1 0 0 0 1 0

  RESULT:     8 9 10 7 3 4 5 6 1 2

  ⍝ ALTERNATE IMPLEMENTATION, FOR NUMBERS > 0:

        V←V[⍋V++\P×⌈/V]

  ⍝ THE SORT INDICES ¨WITHIN EACH FIELD¨ CAN BE
  ⍝  GENERATED VIA:

        I←⎕IO+S[⍋(+\P)[S←⍋V]]-⌈\P×⍳⍴P              **

  RESULT:     3 2 1 1 4 3 2 1 2 1      [⎕IO←1]
```

** ADAPTED FROM (2) *BOOLEAN FUNCTIONS AND TECHNIQUES*

(CONTD ON NEXT PAGE)

SORTING A VECTOR V

SORTING A NUMERIC VECTOR

SORTING THE ELEMENTS OF V WITHIN THE FIELDS OF V (*CONTD*)

IN DESCENDING ORDER

```
V←V[S[⍋(+\P)[S←⍒V]]]                         **
```

EXAMPLE:
```
V←1 2 3 4 5 6 7 8 9 10
P←1 0 0 1 1 0 0 0 1 0
```

RESULT: 3 2 1 4 8 7 6 5 10 9

```
⍝ ALTERNATE IMPLEMENTATION, FOR NUMBERS > 0:

V←V[⍒V++\P×-⌈/V]

⍝ LIKEWISE THE SORT INDICES "WITHIN THE FIELDS" ARE:

I←⎕IO+S[⍋(+\P)[S←⍒V]]-⌈\P×⍳⍴P                **
```

RESULT: 2 1 0 0 3 2 1 0 1 0 [⎕IO←0]

** *ADAPTED FROM* (2) *BOOLEAN FUNCTIONS AND TECHNIQUES*

SORTING A VECTOR V

SORTING A NUMERIC VECTOR

DETERMINING THE RANK-ORDER R OF THE ELEMENTS OF V

ASCENDING RANK-ORDER

```
R←⍋⍋V
```

```
⍝ ρR ←→ ρV
⍝ R[I] IS THE RANK OF V[I] IN V, I.E., THE INDEX
⍝  OF V[I] IN A SORTED V

⍝ NOTE:  THIS IS A "KERNEL" APL OPERATION
```

EXAMPLE:

```
V← 148 149 152 157 153 160 143
V←V, 137 146 151 155 147 150 145
```

RESULT: 6 7 10 13 11 14 2 1 4 9 12 5 8 3

```
⍝ ALTERNATE IMPLEMENTATION:

R←V[⍋V]⍳V
```

DESCENDING RANK-ORDER

```
R←⍋⍒V
```

RESULT: 9 8 5 2 4 1 13 14 11 6 3 10 7 12

```
⍝ ALTERNATE IMPLEMENTATION:

R←V[⍒V]⍳V
```

SORTING A VECTOR V

SORTING A CHARACTER VECTOR

```
⍝ NOTES:

⍝    1. NON-NUMERIC DATA MUST BE ENCODED AS NUMBERS IN ORDER
⍝         TO BE SORTED.

⍝    2. THE NUMERIC SORT OPERATIONS CAN BE MODIFIED FOR
⍝         SORTING A CHARACTER VECTOR V BY REPLACING ¨⍋V¨ WITH
⍝         THE ¨KERNEL¨ APL OPERATION ¨⍋SSEQ⍳V¨ (SEE BELOW
⍝         FOR POSSIBLE ¨SSEQ¨).  LIKEWISE FOR ⍒.

⍝    3. THE SORT SEQUENCE (SSEQ) IS USER-DEFINED.  ¨BLANK¨
⍝         MUST BE POSITIONED WHERE APPROPRIATE.

⍝    4. CAUTION:  ⎕AV IS SYSTEM DEPENDENT.  THE FOLLOWING
⍝          SORT SEQUENCES WITH ⎕AV ARE FOR APLSV.


⍝ POSSIBLE SORT SEQUENCES (SSEQ):

⍝    SSEQ←'ABCD...WXYZ∆ 0123456789'  -  EXPLICIT DEFINITION
⍝    SSEQ←⎕AV  -  SYSTEM DEFINITION:  LETTERS (PLAIN AND
⍝                                UNDERSCORED), NUMBERS, BLANK
⍝    SSEQ←' ',⎕AV  -  BLANK, LETTERS, NUMBERS
⍝    SSEQ←(64↑86↓⎕AV),'⎕'  -  LETTERS, NUMBERS, SPECIAL CHARS
⍝                                          (SEQUENCE FOR LABELS)

⍝  OR ANY OTHER SEQUENCE OF ANY CHARACTERS.
```

MOVING CHARACTERS NOT IN THE SORT SEQUENCE TO END OF V

```
V←V[⍋SSEQ⍳V]
```

EXAMPLE:
```
V←'PROBLEM DEFINITION:   (I÷J)+C⌈52'
SSEQ←'ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789'
```

RESULT:
```
BCDEEFIIIIJLMNNOOPRT    25:(÷)+⌈
```

```
⍝ NOTE:  SUBSTITUTE ¨⍒¨ FOR ¨⍋¨ TO OBTAIN A DESCENDING SORT
```

(CONTD ON NEXT PAGE)

SORTING A VECTOR V

SORTING A CHARACTER VECTOR (CONTD)

DELETING CHARACTERS NOT IN THE SORT SEQUENCE

```
V←V[(I[J]<⎕IO+⍴SSEQ)/J←⍋I←SSEQ⍳V]
```

RESULT: BCDEEFIIIIJLMNNOOPRT 25

```
⍝ ALTERNATE IMPLEMENTATION:

V←V[⍋SSEQ⍳V←(V∊SSEQ)/V]
```

SORTING A VECTOR V

MERGING 2 NUMERIC VECTORS IN ASCENDING ORDER

```
V←V[⍋V←V1,V2]
```

EXAMPLE:

```
V1←53 25 98 33 86
V2←42 79 17 ¯101 64
```

RESULT:

```
¯17 25 33 42 53 64 79 86 98 ¯101
```

```
⍝ NOTE:  SUBSTITUTE "⍒" FOR "⍋" TO OBTAIN A DESCENDING
⍝         MERGE
```

SORTING A VECTOR V

MERGING 2 VECTORS USING THE PATTERN IN LOGICAL VECTOR L

```
V←(V1,V2)[⍋⍒L]
```

```
⍝ 1'S IN L DESIGNATE POSITIONS FOR ELEMENTS OF V1,
⍝    0'S FOR V2
⍝ ⍴L ←→ ⍴V1,V2
```

EXAMPLE: V1←'ACDG'
              V2←'BEF'
              L←1 0 1 1 0 0 1

RESULT: ABCDEFG

```
⍝ ALTERNATE IMPLEMENTATIONS:
```

```
V←(V2,V1)[⍋⍋L]
```

```
V←L\V1
V[(~L)/⍳⍴V]←V2          **
```

** ADAPTED FROM (1) *THE APL IDIOM LIST*

SORTING A VECTOR V

MERGING 2 VECTORS OF SAME LENGTH IN ALTERNATING POSITIONS

```
V3←,V1,[⎕IO+0.5] V2

        ⎕IO←1:  V3←,V1,[1.5] V2
        ⎕IO←0:  V3←,V1,[0.5] V2
```

```
⍝ NOTE:  AXIS-NUMBER DECIMAL-PART IS A DECIMAL VALUE
⍝   BETWEEN 0 AND 1;  "0.5" SHOWN HERE
```

EXAMPLE: V1←1 3 5 7 9
V2←2 4 6 8 10

RESULT: 1 2 3 4 5 6 7 8 9 10

COMPUTING VALUES FROM A VECTOR V

DETERMINING THE WIDTHS W OF THE FIELDS OF V

WHEN THERE ARE NO FIELD DELIMITERS

```
⍝ FROM FIELDS OF IDENTICAL ELEMENTS

        W←I-¯1↓0,I←(~⎕IO)+(((1↓V)≠¯1↓V),1)/⍳⍴V

⍝ ¨I¨ CONTAINS THE ORIGIN-1 INDICES OF THE LAST
⍝   ELEMENTS IN EACH FIELD

⍝ NOTE:  MAY HAVE TO SORT V FIRST TO PRODUCE THE FIELDS
```

EXAMPLE: V←'BBADDDDCCC'

RESULT: 2 1 4 3

WHEN THE FIELD DELIMITER IS Q

```
        W←(I,1+⍴V)-1+0,I←(~⎕IO)+(V=Q)/⍳⍴V

⍝ ASSUMES THAT V CONTAINS NO TRAILING DELIMITERS Q
⍝  (UNLESS THERE IS AN ¨EMPTY¨ FIELD)
```

EXAMPLE: VECTOR OF FIELDS DELIMITED BY COMMAS

```
V←'KR235,RT1,32,TEST VARIATIONS,,9/27/78'
Q←','
```

RESULT: 5 3 2 15 0 7

```
⍝ ALTERNATE IMPLEMENTATIONS:

        W←I-1+¯1↓0,I←(~⎕IO)+((V=Q),1)/⍳1+⍴V

        W←I-1+¯1↓0,I←(~⎕IO)+(U=Q)/⍳⍴U←V,Q
```

COMPUTING VALUES FROM A VECTOR V

SUMMING SETS OF ELEMENTS OF V

SETS OF ELEMENTS SORTED INTO FIELDS

```
⍝ SETS OF IDENTICAL ELEMENTS

        S←W-¯1↓0,W←(V≠1↓V,0)/+\V

⍝ CAUTION:  ANY FIELD DELIMITERS MUST BE DELETED FROM V

EXAMPLE:    V←2 2 5 5 5 4 1 3 3 3

RESULT:     4 15 4 1 9

⍝ TO SUM THE ELEMENTS OF ANOTHER VECTOR "X" WHOSE
⍝  ELEMENTS CORRESPOND TO V, SUBSTITUTE "+\X" FOR "+\V"

⍝ ALTERNATE IMPLEMENTATIONS:

        S←W-¯1↓0,W←(((1↓V)≠¯1↓V),1)/+\V

        S←(V≠1↓V,0)/V×+/V∘.=V
```

COMPUTING VALUES FROM A VECTOR V

SUMMING SETS OF ELEMENTS OF V

SETS OF ELEMENTS NOT IN FIELDS

TYPE 1: *SETS OF IDENTICAL ELEMENTS*

```
S←V+.×V∘.=((⍳⍴V)=V⍳V)/V          **
```

EXAMPLE: V←2 5 4 1 3 2 5 3 5 3

RESULT: 4 15 4 1 9

TYPE 2: *SETS DEFINED BY VECTOR I*

```
S←V+.×I∘.=((⍳⍴I)=I⍳I)/I          **
```

```
⍝ VECTOR ¨I¨ DEFINES THE SETS IN V BY IDENTICAL ELEMENTS.
⍝  ALL ELEMENTS OF V WHOSE CORRESPONDING ELEMENTS IN I
⍝  ARE THE SAME COMPOSE A SET;    ⍴V ←→ ⍴I

⍝ ⍴S ←→ THE NUMBER OF SETS DEFINED BY I

⍝ IF ¨I¨ CONSISTS OF POSITIVE INTEGERS, AND ALL INTEGERS
⍝  OF ¨⍳⌈/I¨ OCCUR IN I, THEN THIS TECHNIQUE CAN BE
⍝  SIMPLIFIED TO:
```

```
S←V+.×I∘.=⍳⌈/I                   **
```

EXAMPLE: V← 72 78 94 83 85 83 76 91 75
I← 22 44 11 55 11 22 55 55 22

RESULT: 230 78 179 250

** *ADAPTED FROM* (1) *THE APL IDIOM LIST*

COMPUTING VALUES FROM A VECTOR V

DETERMINING THE PAIRWISE DIFFERENCES D IN V

```
D←(1↓V)-¯1↓V
```

⍝ ρD ←→ (ρV)-1

EXAMPLE: V←1 4 9 16 25 36 49 64

RESULT: 3 5 7 9 11 13 15

⍝ *ALTERNATE IMPLEMENTATION*:

```
D←1↓V-¯1⌽V
```

⍝ *TO INCLUDE THE 1ST ELEMENT OF V IN THE RESULT* [ρD ←→ ρV]:

```
D←V-¯1↓0,V
```

RESULT: 1 3 5 7 9 11 13 15

⍝ *NOTE*: *THESE ARE "KERNEL" APL OPERATIONS*

COMPUTING VALUES FROM A VECTOR V

COMPUTING THE AVERAGE OF THE VALUES IN V

```
AVG←(+/V)÷1⌈ρ,V          **
```

⍝ *WILL PROCESS A SCALAR AND AN EMPTY VECTOR* (0 = ρ*V*)

EXAMPLE: *V*←86 81 92 73 68 89

RESULT: 81.5

** *ADAPTED FROM* (1) *THE APL IDIOM LIST*

COMPUTING VALUES FROM A VECTOR V

PERFORMING ARITHMETIC SCAN OPERATIONS ON V

```
⍝ THE SIGNIFICANCE OF CERTAIN SCAN OPERATIONS IS AS
⍝  FOLLOWS [ADAPTED FROM (1) THE APL IDIOM LIST]:
```

EXAMPLE: V←1 3 0 0 5 1 0 2

```
+\V   =  PROGRESSIVE SUM:        1 4 4 4 9 10 10 12

×\V   =  PROGRESSIVE PRODUCT:  1 3 0 0 0 0 0 0

⌈\V   =  PROGRESSIVE MAXIMA:   1 3 3 3 5 5 5 5

⌊\V   =  PROGRESSIVE MINIMA:   1 1 0 0 0 0 0 0

⍝ SPECIAL CASE:

-\⍳N  =  ALTERNATING SERIES: [N←7]  1 ¯1 2 ¯2 3 ¯3 4


⍝ ALTERNATE IMPLEMENTATIONS FOR "PLUS-SCAN", WITHOUT
⍝   USING THE SCAN OPERATOR "+\":

       S←V+.×I∘.≤I←⍳⍴V

       D←(-⍴T←-⍴S←V)↑¯1
      LP:S←S+T↑D↓S
       →(T<¯1↑D←2×D)/LP         **
```

** ADAPTED FROM (3) USE AND MISUSE OF APL

COMPUTING VALUES FROM A VECTOR V

PERFORMING LOGICAL SCAN OPERATIONS ON V

```
⍝ THE SIGNIFICANCE OF CERTAIN SCAN OPERATIONS IS AS
⍝  FOLLOWS [ADAPTED FROM (1) THE APL IDIOM LIST]:
```

EXAMPLE: V←0 0 0 0 1 0 1 1 0 1 0

∨\V   = ALL 1'S STARTING WITH THE FIRST (LEFTMOST) 1:

    0 0 0 0 1 1 1 1 1 1 1

<\V   = ALL 0'S EXCEPT THE FIRST (LEFTMOST) 1:

    0 0 0 0 1 0 0 0 0 0 0

≥\V   = ALTERNATING 0'S AND 1'S UNTIL FIRST 1; THEN ALL SAME AS LAST DIGIT:

    0 1 0 1 1 1 1 1 1 1 1

=\V   = ALTERNATING FIELDS OF 1'S AND 0'S, CHANGING TO THE COMPLEMENT AS OF EACH 0. FIRST FIELD OF 0'S OCCURS STARTING WITH FIRST 0 [0'S IN V DESIGNATE 1ST ELEMENTS OF FIELDS IN RESULT]:

    0 1 0 1 1 0 0 0 1 1 0

EXAMPLE: V←1 1 1 0 0 1 0 0 0 1 0 1 0

∧\V   = ALL 0'S STARTING WITH THE FIRST (LEFTMOST) 0:

    1 1 1 0 0 0 0 0 0 0 0 0 0

(CONTD ON NEXT PAGE)

COMPUTING VALUES FROM A VECTOR V

PERFORMING LOGICAL SCAN OPERATIONS ON V (*CONTD*)

EXAMPLE: *V*←1 1 1 0 0 1 0 0 0 1 0 1 0 (*CONTD*)

≤*V* = *ALL* 1'*S EXCEPT THE FIRST* (*LEFTMOST*) 0:

1 1 1 0 1 1 1 1 1 1 1 1 1

>*V* = *ALTERNATING* 1'*S AND* 0'*S UNTIL FIRST* 0; *THEN ALL SAME AS LAST DIGIT*:

1 0 1 1 1 1 1 1 1 1 1 1 1

≠*V* = *ALTERNATING FIELDS OF* 1'*S AND* 0'*S*, *CHANGING TO THE COMPLEMENT AS OF EACH* 1. *FIRST FIELD OF* 1'*S OCCURS STARTING WITH FIRST* 1 [1'*S IN V DESIGNATE* 1*ST ELEMENTS OF FIELDS IN RESULT*]:

1 0 1 1 1 0 0 0 0 1 1 0 0

```
⍝ REPETITIVE EXECUTION OF "≠\" ON A LOGICAL VECTOR
⍝  WHICH INITIALLY HAS ALL 1'S WILL PRODUCE THE
⍝  FOLLOWING BIT-PATTERNS:      **

        1111111111111111    oooooooooooooooo
        1010101010101010    o o o o o o o o
        1100110011001100    oo  oo  oo  oo
        1000100010001000    o   o   o   o
        1111000011110000    oooo    oooo
        1010000010100000    o o     o o
        1100000011000000    oo      oo
        1000000010000000    o       o
        1111111100000000    oooooooo
        1010101000000000    o o o o
        1100110000000000    oo  oo
        1000100000000000    o   o
        1111000000000000    oooo
        1010000000000000    o o
        1100000000000000    oo
        1000000000000000    o
```

** *ADAPTED FROM* (2) ***BOOLEAN FUNCTIONS AND TECHNIQUES***

REPLACING ELEMENTS OF A VECTOR V

REPLACING ELEMENTS SELECTED BY A TEST

WITH ELEMENT Q

```
V[( TEST )/⍳⍴V]←Q
```

```
⍝ THE ¨TEST¨ MUST PRODUCE A BOOLEAN RESULT (E.G.,
⍝  USING A LOGICAL OR RELATIONAL EXPRESSION) WHICH
⍝  IS THE SAME LENGTH AS V

⍝ IF 2≤⍴Q, Q MUST CONTAIN THE SAME NUMBER OF ELEMENTS
⍝  AS THOSE TO BE REPLACED IN V
```

EXAMPLE:
```
V←42 ¯75 0 ¯3 27 0 0 ¯59
TEST←V<0
Q←0
```

RESULT:
```
42 0 0 0 27 0 0 0
```

```
⍝ ALTERNATE IMPLEMENTATION:

      V←(⍴V)⍴( TEST )⊖V,[⎕IO-0.5] Q

⍝ Q IS A SCALAR OR A VECTOR WHERE ⍴Q ←→ ⍴V

⍝ WHEN Q IS A VECTOR AND (⍴Q)<⍴V, Q MUST BE EXPANDED
⍝  TO THE LENGTH OF V, E.G., WITH PAD-ELEMENTS:

      V←(⍴V)⍴B⊖V,[⎕IO-0.5] (B← TEST )\Q

⍝ NOTE:  AXIS-NUMBER DECIMAL-PART IS A DECIMAL
⍝   VALUE BETWEEN 0 AND 1;  ¨0.5¨ SHOWN HERE
```

(*CONTD ON NEXT PAGE*)

REPLACING ELEMENTS OF A VECTOR V

REPLACING ELEMENTS SELECTED BY A TEST (*CONTD*)

WITH ELEMENT Q (*CONTD*)

⍝ *ALTERNATE IMPLEMENTATION, FOR NUMERIC VECTORS:*

```
V←(V×~L)+Q×L←TEST
```

EXAMPLE:

```
V←42 ¯75 0 ¯3 27 0 0 ¯59
TEST←V≤0
Q←2
```

RESULT:

```
42 2 2 2 27 2 2 2
```

⍝ *SPECIAL CASE IMPLEMENTATIONS, REPLACING* 0'*S WITH Q*

```
V←V+Q×V=0          **

V←V+(V=0)\Q
```

EXAMPLE:

```
Q←99
```

RESULT:

```
42 ¯75 99 ¯3 27 99 99 ¯59
```

** *ADAPTED FROM* (1) *THE APL IDIOM LIST*

REPLACING ELEMENTS OF A VECTOR V

REPLACING ELEMENTS SELECTED BY A TEST

WITH THE PAD-ELEMENT (0 *OR BLANK*)

```
V←B\(B←~TEST)/V
```

EXAMPLE: *REPLACING '/' WITH BLANKS*

```
V←'12/31/78'
TEST←V='/'
```

RESULT: 12 31 78

```
⍝ NOTE:  THIS TECHNIQUE ALLOWS FURTHER PROCESSING
⍝   OF V ON THE SAME LINE, IN CONTRAST TO
⍝   REPLACEMENT BY INDEXING
```

REPLACING ELEMENTS OF A VECTOR V

REPLACING MULTIPLE OCCURRENCES OF AN OLD SUBSTRING OS WITH A NEW SUBSTRING NS

```
⍝ MUST FIRST FIND THE STARTING INDICES I OF THE
⍝  OCCURRENCES OF THE OLD SUBSTRING (SEE TECHNIQUES
⍝  "FINDING THE STARTING INDICES ...").  THESE
⍝  INDICES MUST BE IN ASCENDING ORDER.

⍝ THEN COMPUTE THE ADJUSTED STARTING INDICES OF THE
⍝  OCCURRENCES OF THE OLD SUBSTRING

      I←(I-⎕IO)+((⍳NI←⍴I)-⎕IO)×-(OL←⍴OS)-⍴NS

      NI←NI+K←⎕IO-1

     LP:→(NI<K←K+1)/0

⍝ SUBSTITUTE THE NEW SUBSTRING IN PLACE OF THE OLD ONE

      V←(P↑V),NS,(OL+P←I[K])↓V

      →LP
```

EXAMPLE:
```
V←'RV←((⍳⍴RV)=RV⍳RV)/RV'
OS←'RV'
NS←'FACTORS'
I←1 8 12 15 19        [⎕IO←1]
```

RESULT:
```
FACTORS←((⍳⍴FACTORS)=FACTORS⍳FACTORS)/FACTORS
```

```
⍝ NOTE:  IF "NS" IS AN EMPTY STRING, THE OLD STRING IS
⍝   DELETED FROM THE VECTOR


⍝ ALTERNATE IMPLEMENTATION:

      D←~(⍳⍴V)∊(I-⎕IO)∘.+⍳⍴OS
      R←((⍳⍴V)∊I)∘.∧(⍴NS)⍴1
      V←(,D,R)/,V,((⍴V),⍴NS)⍴NS
```

DELETING ELEMENTS FROM A VECTOR V

DELETING ALL OCCURRENCES OF ELEMENT Q

```
V←(V≠Q)/V
```

EXAMPLE: *DELETING BLANKS*

```
V←'  M35 ,    M40, 42.5,TEST,    4/23/78'
Q←' '
```

RESULT: M35,M40,42.5,TEST,4/23/78

⍝ *ALTERNATE IMPLEMENTATIONS, WHEN Q IS A VECTOR:*

```
V←(~V∊Q)/V

V←(∧/V∘.≠Q)/V
```

EXAMPLE:

```
V←'M35,M40,42.5,TEST,4/23/78'
Q←',./'
```

RESULT: M35M40425TEST42378

⍝ *NOTE*: "~V∊Q" IS A "KERNEL" APL OPERATION

DELETING ELEMENTS FROM A VECTOR V

DELETING ALL LEADING OCCURRENCES OF ELEMENT Q

```
        V←(((V≠Q)⍳1)-⎕IO)↓V

                ⎕IO←1:  V←(((V≠Q)⍳1)-1)↓V
                ⎕IO←0:  V←((V≠Q)⍳1)↓V

    ⍝ THE COMPLEMENT IS:  V←(((V=Q)⍳0)-⎕IO)↓V

    ⍝ NOTE:  ¨((V≠Q)⍳1)-⎕IO¨ IS A ¨KERNEL¨ APL OPERATION

    EXAMPLE:    DELETING LEADING ZEROES

                V←0 0 0 22 3 0 14 0 0 57 0 0
                Q←0

    RESULT:     22 3 0 14 0 0 57 0 0

    ⍝ ALTERNATE IMPLEMENTATIONS:

        V←(∨\V≠Q)/V

        V←(+/∧\V=Q)↓V

        V←((1↑(V≠Q)/⍳⍴V)-⎕IO)↓V

        V←(((Q=⌽V)⍳1)-1)↓V

    ⍝ NOTE:  IF ¨Q¨ IS A VECTOR OF DIFFERENT ELEMENTS,
    ⍝   REPLACE ¨V≠Q¨ WITH ¨~V∊Q¨ AND ¨Q≠⌽V¨ WITH ¨~(⌽V)∊Q¨
```

DELETING ELEMENTS FROM A VECTOR V

DELETING ALL TRAILING OCCURRENCES OF ELEMENT Q

```
        V←(⎕IO-(Q≠⌽V)⍳1)↓V

               ⎕IO←1:   V←(1-(Q≠⌽V)⍳1)↓V
               ⎕IO←0:   V←(-(Q≠⌽V)⍳1)↓V

   ⍝ THE COMPLEMENT IS:   V←(⎕IO-(Q=⌽V)⍳0)↓V

   ⍝ NOTE:  ¨⎕IO-(Q≠⌽V)⍳1¨ IS A ¨KERNEL¨ APL OPERATION

   EXAMPLE:    DELETING TRAILING ZEROES

               V←0  0  0  22  3  0  14  0  0  57  0  0
               Q←0

   RESULT:     0  0  0  22  3  0  14  0  0  57

   ⍝ ALTERNATE IMPLEMENTATIONS:

        V←(1-(V=Q)⍸1)↓V          **

        V←((¯1↑(V≠Q)/⍳⍴V)+~⎕IO)↑V

        V←(⌽∨\Q≠⌽V)/V

        V←(-+/∧\Q=⌽V)↓V

   ⍝ NOTE:  IF ¨Q¨ IS A VECTOR OF DIFFERENT ELEMENTS,
   ⍝   REPLACE ¨V≠Q¨ WITH ¨~V∊Q¨ AND ¨Q≠⌽V¨ WITH ¨~(⌽V)∊Q¨

   ** ADAPTED FROM (1) THE APL IDIOM LIST
```

DELETING ELEMENTS FROM A VECTOR V

DELETING LEADING AND TRAILING OCCURRENCES OF ELEMENT Q

```
V←((B⍳1)-⎕IO)↓(⎕IO-(⌽B←V≠Q)⍳1)↓V

        ⎕IO←1:  V←((B⍳1)-1)↓(1-(⌽B←V≠Q)⍳1)↓V
        ⎕IO←0:  V←(B⍳1)↓(-(⌽B←V≠Q)⍳1)↓V

⍝ THE COMPLEMENT IS:  V←((B⍳0)-⎕IO)↓(⎕IO-(⌽B←V=Q)⍳0)↓V
```

EXAMPLE: *DELETING ZEROES*

```
V←0 0 0 22 3 0 14 0 0 57 0 0
Q←0
```

RESULT:

```
22 3 0 14 0 0 57
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

V←((1↑I)-⎕IO)↓((¯1↑I←(V≠Q)/⍳⍴V)+~⎕IO)↑V

V←((∨\B)∧⌽∨\⌽B←V≠Q)/V
```

DELETING ELEMENTS FROM A VECTOR V

DELETING REDUNDANT OCCURRENCES OF ELEMENT Q

```
V←(L∨¯1↓1,L←V≠Q)/V
```

```
⍝ REDUNDANT = THE 2ND, 3RD, ETC., CONTIGUOUS OCCURRENCES
```

EXAMPLE: DELETING ASTERISKS

```
V←'**A****BB*CCC***'
Q←'*'
```

RESULT: *A*BB*CCC*

```
⍝ NOTE:  BOTH    V←(L∨1⌽L←V≠Q)/V
⍝        AND     V←(L⍲1⌽L←V=Q)/V    [THE COMPLEMENT]

⍝   DELETE THE TRAILING OCCURRENCES OF Q ENTIRELY IF Q ALSO
⍝   OCCURS AS THE LEADING ELEMENT.  THE LEADING OCCURRENCES
⍝   ARE DELETED WHEN "...¯1⌽..." IS SPECIFIED.
```

RESULT: *A*BB*CCC

```
⍝ NOTE:  L∨¯1↓1,L  AND  L∨1⌽L  [AND THEIR COMPLEMENTS]
⍝   ARE "KERNEL" APL OPERATIONS
```

DELETING ELEMENTS FROM A VECTOR V

DELETING REDUNDANT OCCURRENCES OF ALL ELEMENTS

```
V←(1,(1↓V)≠¯1↓V)/V
```

```
⍝ REDUNDANT = THE 2ND, 3RD, ETC., CONTIGUOUS OCCURRENCES
```

EXAMPLE: `V←'AAABBCBBBAABCCCC'`

RESULT: `ABCBABC`

```
⍝ NOTE:  THIS SELECTS THE 1ST OCCURRENCES OF ELEMENTS
⍝   IN THE FIELDS OF REDUNDANT ELEMENTS.  TO SELECT THE
⍝   LAST OCCURRENCES:
```

```
V←(((1↓V)≠¯1↓V),1)/V
```

DELETING ELEMENTS FROM A VECTOR V

DELETING LEADING, TRAILING, AND REDUNDANT OCCURRENCES OF ELEMENT Q

```
V←(1↑L)↓(L⍲1⌽L←V=Q)/V
```

EXAMPLE: *DELETING BLANKS*

```
V←'    A BC   DEF    GHIJ       '
Q←' '
```

RESULT: *A BC DEF GHIJ*

DELETING ELEMENTS FROM A VECTOR V

DELETING THE ELEMENTS WITH INDICES I

```
V←(~(⍳⍴V)∊I)/V
```

EXAMPLE:
```
V←15 ¯37 0 22 ¯3 ¯19 8
I←2 5 6      [⎕IO←1]
```

RESULT:
```
15 0 22 8
```

⍝ *ALTERNATE IMPLEMENTATIONS*:

```
V←(∧⌿I∘.≠⍳⍴V)/V
```

```
V←(~∨⌿I∘.=⍳⍴V)/V
```

DELETING ELEMENTS FROM A VECTOR V

DELETING A SUBSTRING WITH STARTING INDEX I AND LENGTH L

```
V←(P↑V),(L+P←I-⎕IO)↓V

     ⎕IO←1:  V←(P↑V),(L+P←I-1)↓V
     ⎕IO←0:  V←(I↑V),(L+I)↓V
```

```
EXAMPLE:   V←'ABCDEFGHIJKLMNOP'
           I←8        [⎕IO←1]
           L←4

RESULT:    ABCDEFGLMNOP
```

```
⍝ ALTERNATE IMPLEMENTATIONS:

     V←(⎕IO-I)⌽L↓(I-⎕IO)⌽V

⍝ NOTE:  THE ABOVE IMPLEMENTATION WILL DELETE A SUBSTRING
⍝    "WRAPPED AROUND" THE ENDS OF V

     V←V[(⍳P),((L+P←I-⎕IO)-⍴V)↑⍳⍴V]

     V←(~(⍳⍴V)∊(I-⎕IO)+⍳L)/V

⍝ TO DELETE MULTIPLE SUBSTRINGS WITH SAME LENGTH:

     V←(~(⍳⍴V)∊(I-⎕IO)∘.+⍳L)/V
```

```
EXAMPLE:   I←1 6 11      [⎕IO←0]

RESULT:    AFKP
```

```
⍝ NOTE:  TO DELETE MULTIPLE OCCURRENCES, SEE ALSO THE
⍝    TECHNIQUE "REPLACING MULTIPLE OCCURRENCES ..."
```

DELETING ELEMENTS FROM A VECTOR V

DELETING DUPLICATE OCCURRENCES OF ALL ELEMENTS

```
V←((⍳⍴V)=V⍳V)/V
```

```
⍝ ELEMENTS OF RESULT OCCUR IN SAME ORDER AS THEIR FIRST
⍝  OCCURRENCE IN V
```

EXAMPLE: V←'TENNESSEE'

RESULT: TENS

```
⍝ ALTERNATE IMPLEMENTATION:
```

```
V←((2⍴⎕IO)⍉<\V∘.=V)/V          **
```

```
⍝ ALTERNATE IMPLEMENTATION, ONLY WHEN V IS NUMERIC
```

```
V←(1,(1↓V)≠¯1↓V)/V←V[⍋V]
```

```
⍝ ELEMENTS OF RESULT ARE SORTED IN ASCENDING ORDER
```

EXAMPLE: V←4 3 2 1 3 2 1

RESULT: 1 2 3 4

** ADAPTED FROM (1) *THE APL IDIOM LIST*

(CONTD ON NEXT PAGE)

DELETING ELEMENTS FROM A VECTOR V

DELETING DUPLICATE OCCURRENCES OF ALL ELEMENTS (CONTD)

```
⍝ SPECIAL CASE IMPLEMENTATIONS:

⍝   WHEN V CONTAINS INDICES AND (⍴V)>⌈/V


        B←((⌈/V)+~⎕IO)⍴0
        B[V]←1
        V←B/⍳⍴B


⍝ ELEMENTS OF RESULT ARE IN ASCENDING ORDER

⍝ CAUTION:  THIS IS NOT EFFICIENT IF ⌈/V IS MUCH GREATER
⍝   THAN ⍴V

⍝ CAUTION:  THE INDICES IN V MUST BE GENERATED IN THE
⍝   SAME ORIGIN IN WHICH THE CODE IS TO BE EXECUTED
```

EXAMPLE: V←5 4 4 3 3 3 2 2 1 4 4 4 3 5 5 2 2 2

RESULT: 1 2 3 4 5   [⎕IO←1]

```
⍝ ALTERNATE IMPLEMENTATION:


        V←(N∊V)/N←⍳(⌈/V)+~⎕IO
```

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

APL Programming Guide

READER'S COMMENT FORM

G320-6103-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

____

____

____

What is your occupation? ____________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

G320-6103-0

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

First Class
Permit 40
Armonk
New York

Business Reply Mail

No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 824
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape | Please Do Not Staple | Fold and tape

Cut or Fold Along Line

APL Programming Guide Printed in U.S.A. G320-6103-0

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

G320-6103-0

APL Programming Guide Printed in U.S.A. G320-6103-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601