A FUNCTION DEFINITION OPERATOR

Kenneth E. Iverson
Peter K. Wooster

I.P. Sharp Associates Limited
145 King St W.
Toronto, Canada M5H 1J8
(416) 364-5361

This paper proposes two related extensions to APL: the extension of assignment to allow a name *F* to be assigned to a derived function by an expression of the form *F*←+.×, and the introduction of a dyadic operator ∇ to apply to character arrays *D* and *M* so that *D*∇*M* produces an ambivalent function in which the dyadic case is defined by *D* and the monadic case by *M*.

Before presenting the formal definition of the operator ∇, we will discuss a number of examples that illustrate the main points. Thus:

```
ROOT←'⍵*÷⍺'∇'⍵*÷2'

ROOT2←'⍵*÷⍺'∇'2∆⍵'

FAC←''∇'⍵×∆⍵-1◇1◇→⍵=0'

FAC2←''∇'A←1+B←0 ◇ A←A×B←B+1 ◇ →0,⍵⍴1'

FAC3←''∇'A←1+B←0◇)B◇L:A←A×B←B+1◇→0,⍵⍴L'

FAC4←''∇'→1+⍵=0◇⍵×∆⍵-1◇1◇'

F←''∇'A←⍵◇B←⍵*2⍝OK◇A×B◇'
```

As in the direct definition on which it is based [1], ⍺ and ⍵ denote the left and right arguments. Moreover, ∆ refers to the function being defined, allowing the use of one case in defining the other (as in *ROOT*2), and allowing self-reference in recursive definition (as in *FAC*). Thus, *FAC* is the factorial function, and *ROOT* and *ROOT*2 are equivalent functions such that *N ROOT X* yields the *N*th root of *X*, and *ROOT X* yields the square root of *X*.

The diamond is a delimiter which breaks the vector into segments *S*0, *S*1, etc., referred to by indices beginning with 0. Execution consists of applying ⍎ to certain of these segments, the explicit result of the function being the result of the last segment which yielded a result (in the sense that *Z*←⍎*S* would not have produced a value error).

The sequence of execution is determined by a **sequence control vector**, which is enclosed [2] to form the first element of the line counter ⎕*LC*. It is initially set to *L*,⍳*L* (where *L* is the index of the last segment and where 0-origin is assumed in the use of ⍳); it is reset to the argument of any branch. Thus in executing *FAC*2 3 it is initially set to 2 0 1 and is immediately reset to 0 1 1 1, causing one execution of segment 0 and three of segment 1. *FAC*2 is therefore an iterative definition equivalent to the recursive definition used in *FAC*.

A segment beginning with a right parenthesis is a **scope control segment**; the name or names it contains are exempted from the localization otherwise applied to any name appearing immediately to the left of the assignment arrow in any of the remaining **normal segments** of the particular case (monadic or dyadic). Thus, *FAC*2 and *FAC*3 differ only in that *B* is global in the latter. The indexing of the normal segments is not affected by the insertion of scope control segments.

The examples *FAC*4 and *F* illustrate how a final diamond (which introduces a final empty segment) effectively causes execution to begin with the leading segment; *FAC*4 is therefore equivalent to *FAC*, and *F* is similar to the familiar use of diamonds. A final segment *L*:→⌽⍳*L* would cause the remaining segments to be executed from right to left.

FORMAL DEFINITION OF ∇

The dyadic operator ∇ applies to character vectors or scalars to produce an ambivalent function; the dyadic case is determined by the left argument and the monadic case by the right. The representation used in the arguments of ∇ is defined as follows:

1. The symbols ⍺ and ⍵ denote the left and right arguments respectively; they are given full status as names, but cannot be used to form longer names such as *B*⍺ or ⍵2. Thus ⍺*ABC*⍵ is equivalent to ⍺ *ABC* ⍵. The symbol ∆ refers to the function being defined.

2. Segments of the representation are delimited by diamonds not in quotes. Any segment whose first non-space character is) is a **scope control segment**; the remaining **normal segments** are referred to by indices beginning with zero.

A label in a normal segment becomes a local constant assigned the value of the associated index, but an error occurs if the label has already been localized (as another label, as an argument, or because of assignment). A comment may occur in any segment.

3. Normal segments are executed in a sequence determined by a **sequence control vector** (which enclosed forms the first element of ⎕*LC*); it is initially set to *L*,⍳*L* (where *L* is the index of the last normal segment and 0-origin is assumed for *L*) and is reset to the argument of any branch statement executed. Execution terminates when the sequence control vector is exhausted or when an invalid index is encountered. A non-integer argument of → produces an error.

4. The explicit result of the function (if any) is the explicit result of the last segment executed which produced one (in the sense that *Z*←⍎*S* is valid). Thus the dyadic case of the function ''∇'⍵' has no explicit result.

5. Localization of names is determined independently for the monadic and dyadic cases. Any name (not in quotes) which (except for possible intervening spaces) occurs immediately to the left of an assignment arrow in any one of the normal segments is localized unless it occurs among the names in one of the scope control segments. The names `α`,`ω`, and `∆` are always localized.

6. Any suspension of the function *F*←*A*∇*B* produces the display *F*`[.5]`, followed by *A* or *B* (according to the case in execution), with an inferior caret to mark the point of suspension. A weak interrupt causes interruption only at the completion of a segment, and the caret then appears just to the right of the segment next to be executed. A branch during suspension behaves like a branch in a segment.

7. If *F*←*A*∇*B*, then `⎕CR 'F'` yields a vector of three enclosed vectors, namely, `(<A)`, `(<B)`, `(<'F')`. In any implementation which does not permit an enclosed vector as an element of ⎕*LC*, it will be necessary to introduce another system function (perhaps named ⎕*LCI*) such that ⎕*LCI* yields the (theoretical) value of `>⎕LC[⎕IO]`, and to assign to `⎕LC[⎕IO]` the actual value .5 . This fractional value will prevent unintended use of `→⎕LC` in resuming execution, and agrees with the display defined in item 6 above.

COMPARISON WITH CANONICAL DEFINITION

The major points of comparison between the proposed operator ∇ and the existing function definition primitive ⎕*FX* may be stated as follows:

1. ⎕*FX* produces a named function, the name being embedded in the argument of ⎕*FX*. On the contrary, ∇ produces an unnamed function which may be applied without assigning it a name, or may be given an arbitrary name by simple assignment. The operator ∇ therefore poses less problems of name conflicts in establishing functions.

2. The operator provides for truly independent definitions of the two cases of the function, including independent localization of names. The facilities for localization are otherwise equivalent, although the conventions for specifying it differ. The automatic localization of names assigned values is extremely convenient.

3. Both modes provide essentially the same facilities for iteration control, except that the operator allows statements to be repeated any specified number of times without repeated tests and branches (as illustrated in the function *FAC*2).

4. The operator provides particularly convenient recursive definition. Moreover, it can be implemented such that many recursions (i.e., those in which the function being defined occurs only as the **root** or **last-executed** function in the expression defining it) can actually be realized as more efficient iterative functions.

5. The use of ◇ as the segment delimiter does not conflict with its use in immediate execution or in canonical definition, but does preclude its use in direct definition. This prevents the occurrence within segments of a sequence of parts which is similar to, but subtly different from, the sequence among segments.

The so-called ∇ form of definition is equivalent to ⎕*FX* except that it provides editing facilities as well. Functions for editing the arguments of the ∇ operator can be easily written (using the functions *QFX* and *QCR* of the working model shown in a later section) For example:

```
REVISE←'0 0ρQFXω,0ρω[α]←<EDIT>(ω←QCRω)[α←⎕IO
+1=α]'∇'2∆ω◇1∆ω◇'
```

```
EDIT←''∇'∆((K↑A≠''/'')/K↑ω),(1↓K↓A),(K←+/∧\A
≠'',''))↓ω◇ω◇→0=ρA←⍞,0ρ⎕←ω'

      REVISE 'ROOT'
ω*÷α
    ,+1
ω*÷α+1

ω*÷2
   /,3
ω*÷3

      ROOT 64
4

     1 REVISE 'ROOT'
ω*÷3
   /,2
ω*÷2
```

Editing and display functions can also be written to display or edit the successive segments on separate lines. In particular, if *M* is a matrix of expressions suitable as segments defining a monadic function, then `''∇(,M,'◇')` defines the function.

A WORKING MODEL

The functions and variables shown below define a workspace which permits one to define functions by simply entering expressions of the form *F*←*A*∇*B* as illustrated by the examples shown at the outset. They are written for *SHARP APL* [3] and use facilities (such as trap and enclosed arrays) which may not be available on all systems. However, they should be rather easily adapted to any *APL* system:

```
⎕TRAP←'∆2 E ⎕ER FIX ⎕ER[1+⎕IO;]'


F←DY DEL MON
F←('2',BUILD DY) ON '1',BUILD MON
F←'⍝',('21',DY ON MON) ON F


F←N IS F;I
N←(' '≠,N)/,N
I←'(0≠⎕NC''α'')RUN SELF←''',N,''''
N←⎕FX('R←α',N,'ω;SELF') ON I ON '→0' ON F


CR←QCR N;I
I←1↑ρCR←⎕CR N
→((5>I)∨(I↑'R(→',Iρ'⍝')∨.≠,(I,1)↑CR)ρ0
CR←(<2↓CR[⎕IO+3;]),(<2↓CR[⎕IO+4;]),<,N


N←N QFX CR;B
→(0∊ρCR)ρL1
→((~B⎕<>B←''ρCR),0≠⎕NC 'N')/L1,L0
N←>CR[⎕IO+2]
L0:CR←N IS(>CR[⎕IO]) DEL>CR[1+⎕IO]
→0
L1:N←⎕FX CR
R←α ∆ ω
(0≠⎕NC 'α') RUN SELF
B ω
SL←,ω ◇ ⎕SIGNAL 999
R←BUILD A;G;L;Y;⎕IO;U;T
→(2=ρρR←A)ρ0
R← 1 0 ρ⎕IO←0
→((0∊ρA)∨' '∧.=,A)ρ0
L←LCLS R←>''ρY←EXPS '→B' IN A
G← 0 1 ↓PVTOM ' ⎕ ⍞ α ω ',',' ',>Y[2]
Y←(T←Y∨.≠' ')⌿Y←>Y[1]
U←,';',Y ON L LESS Y ON G
U←U,,'◇',Y,'←',⍕((ρT),1)ρT←T/⍳ρT
R←((U≠' ')/U) ON R
ER←ERRD ER;⎕IO
ER←(SELF,' VALUE ERROR') ON ER[1 2 +⎕IO←0;]

S←EXPS S;T;I;J;M;G
T←(S HAS '◇⍝') PVTOM S←'◇',S,' '
S← 0 1 ↓('⍝'≠T[;⎕IO])⌿T
M←⌈/I←+/⌽∨\⌽S HAS ':'
S←(I-M)⌽S,((ρI),M)ρ' '
G← 0 1 ↓(J←')'=T[;0])⌿T←(+/∧\' '=S)⌽S
S←(<(~J)⌿(0,M)↓S),(<(~J)⌿S[;¯1↓⍳M]),<G

A FIX W
W←(W HAS '←∇') PVTOM W←' ',W
→((' ←∇'∧.= 3 1 ↑W)∧3=1↑ρW)↓0
A←(W ND0 0) IS(⍎1↓W ND0 1) DEL⍎1↓W ND0 2


R←A HAS W
R←(A∊W)∧=\''''≠A
A←A IN W
A←(,(W HAS 1↑A)∘.≠4↑1)/,W,(⌽3,ρW)ρ1⌽'  ',1↓A


A←LCLS A;I
A←⌽ ¯1 1 ↓(A HAS '←') PVTOM A←,'◇',A
A←(+/∧\' '=A)⌽A
I←∧\A∊64↑86↓⎕AV
A←(ρA)ρI\(I←,∨\(⌽I)∧A∊54↑86↓⎕AV)/,A←⌽A
A←(A∨.≠' ')⌿(+/∧\A=' ')⌽A


L←L LESS G;I
→(0∊(ρL),ρG)ρ0
I←(' ',⎕AV)⍋L←G ON L
G←⎕IO+''ρ¯2↑1,ρG
L←L[((I≥G)∧1,∨/L[1↓I;]≠L[¯1↓I;])/I;]


R←A ND0 B;⎕IO
R←A[B+⎕IO←0;]

R←R ON B;I
R←(¯2↑ 1 1 ,ρR)ρR
B←(¯2↑ 1 1 ,ρB)ρB
I←0,¯1↑(ρB)⌈ρR
R←((I⌈ρR)↑R),[⎕IO](I⌈ρB)↑B


R←B PVTOM VEC;T;V
→(0≠⎕NC 'B')ρL0
B←VEC∊1↑,VEC
L0:R←((×R),R←×/ρVEC)ρVEC
→(0∊(ρB),ρVEC)ρ0
V←(1↓T)-¯1↓T←(1,(1↓(ρ,VEC)ρB),1)/⍳1+ρ,VEC
T←V∘.≥(⍳⌈/V)+~⎕IO
R←(ρT)ρ(,T)\VEC


D RUN FN;RUN1
RUN0 ⍝BUILD AND FIX RUN1
RUN1 ⍝RUN1 DOES LOCALIZATION
⍝RUN2 RUNS VECTOR (DIRECT) DEFNS
```

```
RUN0;S;⎕IO
⎕IO←0
FN← 5 1 ↓⎕CR FN
FN← 0 1 ↓((1+D)=⍎,' ',FN[;⍳1])⌿FN
S←FN[0;]⍳'◇'
S←('RUN1 ',S↑FN[0;]) ON '◇',S↓FN[0;]
S←⎕FX S ON 'RUN2'

RUN2;⎕TRAP;⎕ER;SL;SL0
SL←¯1⌽¯1++\(1↑ρFN← 1 0 ↓FN)ρ1
RES:⎕TRAP←'∇6 C →ERR∇999 C →RES'
CONT:→((0∊ρSL)∨∨/SL∊⎕AV)ρ0
→((SL0<0)∨(SL0←''ρSL)≥1↑ρFN)ρ0
SL←1↓SL
R←⍎,FN ND0 SL0
→CONT
ERR:→('RUN2[6] '∨.≠8ρ 1 0 ↓⎕ER)/ER2
→CONT
ER2:SΔRUN2←ER9,0ρ⎕←ERRD ⎕ER
ER9:
```

ITERATIVE EQUIVALENTS OF RECURSION

If in the monadic definition of a function *F* the last element of the argument of a branch is *I*, and if segment *I* is of the form Δ *G* ω, then the recursive use of *F* invoked by the self-reference Δ can be replaced by executing ω←*G* ω followed by →*ISC* *N*, where *ISC*←''∇'¯1⌽¯1++\ωρ1', and *ISC* *N* is the normal initial value of the sequence control vector for the function *F* consisting of *N* segments.

The essential point is that Δ occurs as the **root** function (last to be executed) in the last segment to be executed in *F*. Similar remarks apply to the dyadic case.

We will treat the monadic case further by showing how any definition can be modified (largely by appending segments of the form *NR* ω←*G* ω, where the function *NR* is defined by *NR*←''∇'' and prevents the formation of new explicit results due to the execution of these segments) to produce an equivalent definition that uses iteration instead of recursion wherever possible.

Briefly, a branch of the form →*B* is replaced by the expression →*RS* *RI* *B*, where

```
RI←'ω◇(¯1↓ω),α[0;I],ISC 1ρα◇ →(¯1↑ρα)>I←α[1+
⎕IO←0;]⍳¯1↑¯1,ω←ω×¯1*ω≥1ρα'∇''
```

and where *RS* is a matrix whose second row lists the indices of those **eligible** segments for which iterative substitution is possible (i.e., those of the form Δ *G* ω), and whose first row lists the indices of the corresponding new segments *NR* ω←*G* ω. For example, if

```
F←''∇'Δω-1◇ω-2◇→?2◇Δω-3◇→?4'
```

then the eligible segments are 0 3, the corresponding added segments are 5 6, and the value of *RS* would be 2 2 ρ 5 6 0 3. The corresponding equivalent function *G* is therefore given by:

```
G←''∇'Δω-1◇ω-2◇→RS RI ?2◇Δω-3◇→RS RI ?4 ◇NRω
←ω-1 ◇NRω←ω-3 ◇→ISC 1ρRS←2 2ρ5 6 0 3'
```

The final segment of *G* (which is invoked first) establishes the appropriate value of *RS* and then sets the sequence control vector to the initial value appropriate to *F*, that is, to the initial part of *G*.

A **compile** function which can be used in the form *'G' COMPILE 'F'* to define *G* as the iterative form of *F*, and in the form *COMPILE 'F'* to reassign to *F* its iterative form, may be defined (using as subfunctions some of those used in the model given earlier for the operator ∇) as follows:

```
ALGN
  (α-M)⌽ω,((⍴α),M←⌈/α)⍴' '

BLK1
 (~(1⌽I)∧I←ωHAS' ')/ω

BRFIX
 (ω,(+/B)⍴'RS RI ')[⍋(⍳⍴ω),(B←6×('R'≠1⌽ω)∧I←
ωHAS'→')/⍳⍴ω]

CDY
 ω⋄BLK1(BRFIXω),S EXTRAD T⋄→~0∊⍴T←(S≠0)⌿(S←
TAILS T)ALGN T←>''⍴EXPS ω

CMON
 ω⋄BLK1(BRFIXω),S EXTRAM T⋄→~0∊⍴T←(S←'Δ'=T[;
⎕IO])⌿T←>''⍴EXPS ω

COMPILE
 αQFX(<CDY>ω[0]),<CMON>(ω←QCRω)[1+⎕IO←0]
 ωΔω

DYIT
 ((S,15)⍴'NRω←>(⍳0)⍴⌽(<α←'),((S,⌈/α)↑ω ),(((
S←1↑⍴ω),3)⍴'),<'),((0,1+⌈/α)↓ω ),'⋄'

ESG
 '→ISC 1⍴RS←2 ',(⍕+/ω),'⍴',⍕((⍴ω)+⍳+/ω),ω/⍳⍴ω

EXTRAD
 '⋄',(,(⌈/α)DYIT ω),ESG α≠0

EXTRAM
 '⋄',(,(((1↑⍴ω),4)⍴'NRω←'),0 1↓ω,'⋄'),ESG α

PAREN
 (×ω)×1+ω←+/∧\0≠+\(¯1×ω HAS ')')+ω HAS '('

TAILS
 R←((⍴S),2)↑(S←PAREN ω)⌽ω ⋄ S×(R[;0]='Δ')∧~R
[;1]∊54↑86↓⎕AV⋄

ISC
 ¯1⌽¯1++\ω⍴1

NR←''∇''

RI
 ω⋄(¯1↓ω),α[0;I],ISC 1⍴α⋄→(¯1↑⍴α)>I←α[1+⎕IO←
0;]⍳¯1↑¯1,ω←ω×¯1*ω≥1⍴α
```

One simplifying assumption is made in this model, namely, that the left argument of Δ in the dyadic case is always enclosed in (possibly redundant) parentheses.

It is interesting to note that the function *FAC* does not permit compilation into iterative form but that, because the root function × is associative, one can define an equivalent function that does. Thus:

```
F←'(α×ω)Δω-1⋄α⋄→ω=0'∇'1Δω'
```

The function *G* resulting from *'G' COMPILE 'F'* is defined by:

```
G←'(α×ω)Δω-1⋄α⋄→RS RI ω=0 ⋄NRω←>(⍳0)⍴⌽(<α←(α
×ω)),<ω-1 ⋄→ISC 1⍴RS←2 1⍴3 0'∇'1Δω'
```

ACKNOWLEDGMENTS

We are pleased to acknowledge the helpful criticism and advice of our colleagues at I.P. Sharp Associates, particularly Bob Bernecky, Roland Pesch, Doug Forkes, Arthur Whitney, and Jerry Cudeck.

REFERENCES

1. Iverson, K.E., **Elementary Analysis**, APL Press, 1976.

2. Bernecky, Bob and Iverson, K.E., Operators and Enclosed Arrays, **APL Users Meeting**, I.P. Sharp Associates, 1980.

3. Berry, P.C., **Sharp APL Reference Manual**, I.P. Sharp Associates, 1979.