MSPM SECTION BZ.10.07

Identification

APL Operators and Requests

Purpose

This component of the APL interpreter contains the routines which actually implement each of the many different primitive functions, operators, and system requests of the APL language.

Environment

The request processor is called upon from three different places:

- To process a request given as a command argument to the APL command, the processor is called from the parser's initialization sequence. The arguments of the request will have been collected as a character string by the parser.
- To process a request entered in the normal way, the processor will be called from the lexical analyzer when it detects the initial right parenthesis. The lexical analyzer will make available the remainder of the line.
- To process a request programmably called by a function reference in an APL statement being interpreted, the processor will be called from the action coding of the reduction rule detecting the call. The right-hand operand of the function reference, which must be a character array, is made available to the request processor.

In each case, the arguments of the request are simply passed to the request processor as character strings.

The request processor validates the request name and dispatches to its handler. Each handler is responsible for tearing down its own arguments.

The operator processors are all called only from the action coding of reduction rules. Each rule which makes a promotion to VAL, EXP, or LIST will, in general, call an operator routine to perform the operation semantically implied by the promotion. The operator routine sees the parser's reduction stack as it is before the promotion, so that the argument tokens and pointers to the values they represent are accessible.

Each operator routine individually checks the suitablity of its arguments as to rank and type, signalling RANK, LENGTH, DOMAIN errors as appropriate. Based on the input arguments, an estimate is made of the type and size of the result, and storage is allocated in the value segment to accept it. Then the routine actually performs the operation. Finally, the storage occupied by the operands is freed, and the operator routine returns to the parser with the pointer to the result value. The parser will restructure the reductions stack to reflect the promotion, inserting the new value pointer into the new stack top token.

In some cases, the estimated size of result can be too low (e.g. integer + integer is estimated to yield integer, but in fact the sum could overflow integer capacity and force conversion to float). In these cases, the operator routine carefully checks for overflow as it is performing the operation. If overflow occurs, a new result space is allocated, previously calculated results are copied from the old result space and converted, the old result space is junked, and the operation continues from the point of overflow detection, now storing into the new result space.

In all cases, the states of value allocations are recorded in standardized places, so that storage can be properly freed in the event of an interrupt.

Operator Actions

The operators are triggered by reduction rules which promote syntactic categories. The operation performed for each such

reduction rule are summarized below, each listed below the rule which triggers it.

The operations numbered 1 through 4, 9 through 21, and 58 are mostly concerned with proper maintenance of the reductions stack, and do not have much significance to the APL language itself. The operations numbered 22 through 57, on the other hand, are responsible directly for implementation of the user-accessible APL primitives. The operations numbered 5 through 8 are executed at the completion of the interpretion of a line; hence, they shed light on the final disposition of a line.

1. val con $
   - Copy the value located in the CON token over into the value area. Set its usage count to 1.
   - Set value pointer of VAL token to point to it.

2. val zfn $
   - Note that the usage bead pointed to by the usage pointer in the ZFN token must be valid, because no function calls have occured yet.
   - An F-frame is spawned onto the stack.
   - The F-frame is hooked to the function usage bead. The usage count in the bead need not change, since the reference in the ZFN token is disappearing.
   - If the function allows a result, enter the local name into the symbol table, and record its usage pointer in the frame's local name list.
   - If the function has local variables, enter them into the symbol table, and record their usage pointers in the frame's local name list.
   - Scan the function's lexed lines for labels, enter them into the symbol table with their values, and record their usage pointers in the frame's local name list.
   - Go to the top of the parse main loop.
   - When the function returns, change the ZFN token to a VAL token.

- Place the returned value pointer in the VAL token.

3. val "qq $
   - Read the next line on the current input stream.
   - Create a character value in the value area, giving it a usage count of 1.
   - Hook this VAL token to it.

4. val var $
   - The usage bead pointed to by the usage pointer of this token must still be valid.
   - Move the value pointer from the usage bead to the VAL token.
   - Signal VALUE ERROR if the pointer is null (distinguish carefully between a null pointer and a null value).
   - Add one to the value's usage count; subtract one from the variable's usage count.

5. S bol } eol $
   - If this is an F-frame, SYNTAX ERROR on }.
   - If there is no previous frame, SYNTAX ERROR on }.
   - Back up to the previous frame, deallocating the current frame.
     This involves:
     - If the frame has a parse stack, cycle through it, deallocating values pointed at by VAL, EXP, and LIST beads.
     - If the frame has a local-name-list, cycle through the list deallocating the usage beads (and, if the usage bead has a non-null value pointer, deallocate the value bead too).
   - If the current frame now is not an S-frame, go back to the previous step.
   - Go to the top of the parser's main loop.

6. S bol } list eol $

- If the current frame is an E-frame, SYNTAX ERROR on }.
- If the current frame is an S-frame,
  - If there is no previous frame, SYNTAX ERROR on }.
  - If the previous frame is not an F-frame, SYNTAX ERROR on }.
  - Deallocate the current frame and restore the previous frame.
- If the list has more than one member, print all members on the current output stream.
- If the list consists of exactly one non-existent value (N.B.: value pointer is null, not null value), then VALUE ERROR on it.
- Unless the first expression of the list has an integer value, DOMAIN ERROR on }.
- If trace is enabled for this line, print the trace message.
- Set the new current line number.
- Return to the top of the parser main loop.

7. S bol list eol $

- If the list has more than one member, or if the PRINT indicator is on, print all values.
- I f trace is enabled for this line, print the trace message.
- If the current frame is an E-frame,
  - VALUE ERROR on LIST unless the first expression of the list has a value.
  - Return a pointer to the first expression of the list as the value of the evaluated input.
  - Deallocate storage occupied by the remainder of the list.
  - Deallocate the current frame and return to the processing of the evaluated-input call in the preceding frame.

- Otherwise, return the entire list to storage.
- If the current frame is an F-frame, bump the line number.
- Go to the top of the parser main loop.

8. S bol eol $
   - If trace is enabled on this line, print the trace message.
   - If the current frame is an F-frame, bump the line number.
   - Return to the top of the parser main loop.

9. cop / $
   - Place '/' in the third word of the COP token.

10. cop backslash $
    - Place a backslash in the third word of the COP token.

11. val (exp) $
    - Copy the value pointer from the EXP token to the VAL token.
    - Signal VALUE ERROR on EXP if the value pointer was null.
    - There is no need to set the "source" pointer of the VAL token, since no further value error can occur.

12. val var [ ] $
    - Ignore the brackets. This is treated the same as action 4.

13. val var [list] $
    - Has the usage bead pointed to by the VAR token been junked? If so,
      - Deallocate the junked usage bead.
      - Search the symbol table for a new referent for the variable name.

- If not found, SYNTAX ERROR on VAR.
- If found, but now function or group, SYNTAX ERROR on VAR.
- Hook VAR token to new variable usage bead.

- If value pointer in usage bead is null, VALUE ERROR on VAR.
- Copy the value pointer to the VAL token. Bump the value's usage count; decrement the variable's usage count.
- Now treat as action 15, below.

14. val val [ ] $

- Ignore the brackets. Leave the value pointer in the VAL token untouched.
- If the value pointer is null, signal VALUE ERROR on VAL.

15. val val [list] $

- VALUE ERROR on VAL if its value pointer is null.
- VALUE ERROR on LIST if it consists of exactly one null expression.
- Allocate space to hold the sub-array result: the type of the result is the type of VAL; the rank of the result is the sum of the ranks of the LIST elements; the number of the result is the product of the numbers of the list elements.
- Extract the result.
- Replace the value pointer in the VAL bead, de-allocating the old value.
- Deallocate LIST values.

16. list list ; $

- VALUE ERROR on LIST if it consists of exactly one expression with a null value pointer.
- Add an expression of null value pointer to the list.

17. list ; $

- Create a list consisting of exactly two expressions, both with null value pointers.
- Set the "source" pointer of the LIST token equal to the "source" pointer of the semicolon token.

18. val "qu $

- Spawn an E-frame on top of the current state frame.
- Go to the top of the parser main loop.
- When the E-frame returns, place the returned value pointer into the VAL bead (it has already been checked for VALUE ERROR).

19. exp val $

- Copy the value pointer from the VAL token to the EXP token.
- Set the PRINT switch ON, unless the value pointer is null, in which case set it OFF.

20. list list ; exp $

- VALUE ERROR on EXP if its value pointer is null.
- VALUE ERROR on LIST if it consists of exactly one expression having a null value pointer.
- Add the expression to the list.

21. list ; exp $

- VALUE ERROR on EXP if its value pointer is null.
- Create a list of two elements, the first having null value pointer, the second having the value pointer from EXP.
- Set "source" pointer in LIST token from "source" pointer of semicolon token.

22. exp "qu { exp $

- Signal VALUE ERROR on EXP if its value pointer is null.

- Copy the value pointer into the new EXP node.
- Print the value onto the current output stream.
- Turn the PRINT switch OFF.

23. exp var { exp $

- Signal VALUE ERROR on EXP if its value pointer is null.
- If the usage bead pointed to by the VAR token's usage pointer has been junked:
  - Deallocate the junked usage bead.
  - Search the symbol table for a new referent for the variable name.
  - If none found, SYNTAX ERROR on VAR.
  - If found, but not a variable, SYNTAX ERROR on VAR.
  - Hook up to new usage bead.
- If the value pointer in the usage bead is non-null, deallocate the value.
- Copy the expression's value pointer to the usage bead and also to the new EXP token. Bump the usage count of the value by one.
- Decrease the variable's usage count by one.
- Turn the PRINT switch OFF.

24. exp var [ ] { exp $

- Ignore the brackets. Treat as 23, above.

25. exp var [list] { exp $

- VALUE ERROR on EXP if its value pointer is null.
- VALUE ERROR on LIST if it consists of exactly one member with a null value pointer.
- Verify, as in action 23 above, that the referent of the VAR token has not changed.
- If the value usage count of the present value of the variable VAR is greater than one (the value is being used for some other purpose as well), or if the type of EXP is greater than the type of VAR, then copy

over the value. DOMAIN ERROR on { unless the types of VAR and EXP are both numeric or both character.

- Substitute the new values into the variable's value bead, as directed by the LIST.
- Deallocate the storage used by LIST and EXP.
- Turn the PRINT switch OFF.

26. exp stc { exp $

- VALUE ERROR on EXP if its value pointer is null.
- SYNTAX ERROR on STC unless the function named in it actually exists.
- RANK ERROR on { unless the EXP has a value which is a scalar, a vector, or consists of one element.
- Cycle through the elements of the expression, setting stop and trace control bits on the applicable lines of the function. DOMAIN ERROR on { if a non-integral number is found, or a number for which no line exists.
- Set the PRINT switch OFF.

At this point, some general comments are given which apply to actions 27 through 57 (except 31 and 44, which have to do with function calls). All these are various APL primitive operators which promote to expressions. The actions for them all obey the universal pattern:

- VALUE ERROR is signalled if and EXP or VAL appearing in the construct has a null value pointer.
- RANK, LENGTH, and DOMAIN checks are made by the individual operators, as appropriate. See the individual discussions below.
- The type, rank, and number of the result value is calculated, as appropriate to the individual operator.
- Space is allocated in the value area to hold the result (the type, rank, and number determine how big the storage allocation must be).
- The actual operation is performed.
- If the estimate of the result space is wrong (which can occur only because the result was assumed to be of type

integer, but a result value has occured which overflows integer capacity), the operator will catch the overflow, allocate a new result space (of type float), copy over the previous results converting them to floating, de-allocated the old result space, and resume the operation, working into the new result space.

- Deallocate all VALs and EXPs.
- Save the pointer to the result in the new EXP token.
- Turn the PRINT switch ON.

27. exp val sop exp $

28. exp val lop exp $

29.1 exp val ? exp $

- These three cases are treated alike.
- RANK ERROR on the operator unless the operand ranks are identical (except that a scalar on either side matches any rank by extension).
- LENGTH ERROR on the operator unless the rhos of the operands are identical.
- The rank and number of the result is the common rank and number of the operands. The type of the result depends upon the type of the operands as well as the particular operator. The chart on the next page shows, for each operator, the type of the result as a function of the types of the operands.

29.2 exp val % exp $

- RANK ERROR unless VAL is a scalar or a vector.
- The type of the answer is the type of EXP: the rank of the answer is the rho of VAL; the number of the answer is the multiplication-reduction of VAL.
- DOMAIN ERROR if the number of the answer is greater than zero, but the number of EXP is zero, or if VAL is other than numeric non-negative integer valued.

Type of Result of Dyadic Operators As a Function of Type of Left and Right Operands (B=bit, I=integer, F=float, C=character, J=try integer but may go to float, -=DOMAIN ERROR)

operation	BB	BI	BF	BC	IB	II	IF	IC	FB	FI	FF	FC	CB	CI	CF	CC
+ - & "ev	I	J	F	-	J	J	F	-	F	F	F	-	-	-	-	-
÷	B	I	F	-	I	F	F	-	F	F	F	-	-	-	-	-
"ce "fl	B	I	F	-	I	I	F	-	F	F	F	-	-	-	-	-
*	B	B	B	-	I	J	F	-	F	F	F	-	-	-	-	-
"lo	-	-	-	-	B	F	F	-	B	F	F	-	-	-	-	-
| "en	B	I	F	-	B	F	F	-	F	F	F	-	-	-	-	-
!	B	I	F	-	B	J	F	-	B	F	F	-	-	-	-	-
?	B	I	-	-	B	I	-	-	-	-	-	-	-	-	-	-
"ci	F	F	F	-	F	F	F	-	-	-	-	-	-	-	-	-
"an "or	B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
"na "no	B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
< ≤ > ≥	B	B	B	-	B	B	B	-	B	B	B	-	-	-	-	-
= ≠ "ep	B	B	B	-	B	B	B	-	B	B	B	-	-	-	-	B
,	B	I	F	-	I	I	F	-	F	F	F	-	-	-	-	C
% "up "do	B	I	F	C	B	I	F	C	-	-	-	-	-	-	-	-
"rr "tr	B	I	F	C	B	I	F	C	-	-	-	-	-	-	-	-
$	I	I	I	-	I	I	I	-	I	I	I	-	-	-	-	I
/ \	B	I	F	C	-	-	-	-	-	-	-	-	-	-	-	-

Type of Result of Monadic Operators As a Function of Type of Right Operand

operation	B	I	F	C
+	B	I	F	-
-	B	J	F	-
& "gu "gd	I	I	I	-
"lo	B	F	F	-
"ce "fl	B	I	J	-
* "ci	F	F	F	-
! |	B	J	F	-
$?	B	I	-	-
tilde	B	-	-	-
%	I	I	I	I
, "rr "tr	B	I	F	C

29.3 exp val , exp $

- DOMAIN ERROR if VAL and EXP are not both type numeric or both type character.
- The type of the answer is given in the chart. The rank of the answer is one. The number of the answer is the sum of the numbers of VAL and EXP.

29.4 exp val $ exp $

- DOMAIN ERROR if VAL and EXP are not both type numeric or both type character.
- RANK ERROR unless VAL is a scalar or a vector.
- The type of the result is integer. The rank and number of the result are the rank and number of EXP.

29.5 exp val "tr exp $

- RANK ERROR unless the rho of VAL is equal to the rank of EXP, except that a scalar VAL will match a vector EXP.
- DOMAIN ERROR unless VAL is of type numeric, and has values chosen from and exhausting the set $ "ce/VAL.
- The type, rank, and number of the result are the type, rank, and number of EXP.

29.6 exp val "rf exp $

- RANK ERROR unless the rho of VAL is equal to %(^1'doEXP).
- DOMAIN ERROR unless VAL is numeric integer valued.
- The type, rank, and number of the result are the type, rank, and number of EXP.

30.1 exp val "up exp $

- RANK ERROR unless VAL is a scalar or a vector.
- RANK ERROR unless the number of VAL is equal to the rank of EXP.
- DOMAIN ERROR unless ("an/(%EXP)≥|VAL) = 1.
- The type and rank of the result are the type and rank of EXP. The number of the result is the multiplication-

reduction of the magnitude of VAL.

30.2 exp val "do exp $

- RANK ERROR unless VAL is a scalar or a vector.
- RANK ERROR unless the number of VAL is equal to the rank of EXP.
- DOMAIN ERROR unless ("an/(⍴EXP)≥|VAL) = 1.
- The type and rank of the result are the type and rank of EXP. The number of the result is &/(⍴EXP)-|VAL.

30.3 exp val "ep exp $

- DOMAIN ERROR if VAL and EXP are not both type numeric or both type character.
- The rank and number of the result are the rank and number of VAL. The type of the result is bit.

30.4 exp val "ev exp $

- RANK ERROR unless both VAL and EXP are scalars or vectors, and, if both vectors, of the same length.
- DOMAIN ERROR unless both VAL and EXP are numeric.
- The result is a numeric scalar.

30.5 exp val "en exp $

- DOMAIN ERROR unless VAL and EXP are both of type numeric.
- RANK ERROR unless EXP is a scalar and VAL is a scalar or a vector.
- The rank and number of the result are the rank and number of VAL. The type of the result is given in the type chart.

30.6 exp val ⍀ exp $

- DOMAIN ERROR unless VAL is of type bit.
- RANK ERROR unless (+/VAL)=(⍴EXP)⌊1⌋.
- The type and rank of the result are the type and

rank of EXP. The number of the result is (#EXP)+(+/1-VAL)&(#EXP)÷(%EXP)⌈17. Here, #EXP means the number of EXP; i.e., #EXP = &/%EXP.

31. exp val dfn exp $

- VALUE ERROR if either VAL or EXP have null value pointers.
- If the usage bead pointed to by the DFN token has been junked:
  - Deallocate the junked usage bead.
  - Search the symbol table for a new referent for the function name.
  - If not found, SYNTAX ERROR on the DFN token.
  - If found, but not a dyadic function anymore, SYNTAX ERROR on DFN.
  - Hook the DFN token to the new usage bead.
- Spawn an F-frame to the state stack, and hook it to the function's usage bead.
- Place the function arguments in the symbol table with the proper values, and enter their usage pointers into the frame's local name list.
- If the function allows a result, enter the local name into the symbol table, and record its usage pointer in the frame's local name list.
- If the function has local variables, enter them into the sumbol table, and record their usage pointers in the frame's local variable list.
- Scan the function's lexed lines for labels, enter them into the symbol table with their values, and record their usage pointers in the frame's local variable list.
- Go to the top of the parse main loop.
- When the function returns, place the result value pointer in the new EXP token.
- If the value pointer is null then set the PRINT switch OFF; otherwise, set it ON.

32. exp val ⍀ exp $

- DOMAIN ERROR unless VAL is of type bit.
- RANK ERROR unless VAL is a scalar or else a vector of length (⍴EXP)⌈1⌉.
- The type and rank of the result are the type and rank of EXP. The number of the result is (#EXP)&(+/VAL)÷(#VAL).

33.1 exp val / exp $

- DOMAIN ERROR unless VAL is of type bit.
- RANK ERROR unless VAL is a scalar or else a vector of length (⍴EXP)⌈⍴⍴EXP⌉.
- The type and rank of the result are the type and rank of EXP. The number of the result is (#EXP)&(+/VAL)÷(#VAL).

33.2 exp val \ exp $

- DOMAIN ERROR unless VAL is of type bit.
- RANK ERROR unless (+/VAL) is equal to (⍴EXP)⌈⍴⍴EXP⌉.
- The type and rank of the result are the type and rank of EXP. The number of the result is (#EXP)+(+/1-VAL)&(#EXP)÷(⍴EXP)⌈⍴⍴EXP⌉.

34. exp val "rr exp $

- RANK ERROR unless VAL is a scalar or else (⍴⍴VAL)=(⍴⍴EXP)-1 and (⍴VAL)=⍴(⍱1"doEXP).
- DOMAIN ERROR unless VAL is numeric integer valued.
- The type, rank,and number are the type, rank, and number of EXP.

35. exp val sop . sop $
36. exp val sop . lop $
37. exp val lop . sop $
38. exp val lop . lop $

- The type of Vf.gE is the type of (VgE)f(VgE), from the chart.
- LENGTH ERROR unless (⍴VAL)⌈⍴⍴VAL⌉=(⍴EXP)⌈1⌉.

- The rank of the result is the rank of VAL plus the rank of EXP less two.
- The number of the result is (#VAL)&(#EXP)÷(%EXP)⌈1⌋*2.

39. exp val "cc . sop exp $

40. exp val "cc . lop exp $

- The type of V"cc.fE is the type of VfE, from the chart.
- The rank of the result is the sum of the ranks of the operands.
- The number of the result is the product of the ranks of the operands.

41.1 exp val / ⌈exp1⌋ exp2 $

- RANK ERROR unless EXP1 is a scalar.
- DOMAIN ERROR unless EXP1 is a numeric integer $0 \leq$ EXP1-INDEX $<$ (%%EXP2), where INDEX is the index origin, and VAL is of type bit.
- RANK ERROR unless VAL is a scalar or a vector of length (%EXP2)⌈EXP1⌋.
- The type and rank of the result are the type and rank of EXP2.
- The number of the result is (#EXP2)&(+/VAL)÷(#VAL).

41.2 exp val \ ⌈exp1⌋ exp2 $

- DOMAIN ERROR unless EXP1 is a numeric integer $0 \leq$ EXP1-INDEX $<$ (%%EXP2).
- RANK ERROR unless EXP1 is a scalar.
- DOMAIN ERROR unless VAL is of type bit.
- RANK ERROR unless (+/VAL) = (%EXP2) ⌈EXP1⌋.
- The type and rank of the result are the type and rank of EXP2. The number of the result is (#EXP2)+(+/1-VAL)&(#EXP2)÷(%EXP2)⌈EXP1⌋.

42. exp val "rr ⌈exp1⌋ exp2 $

- DOMAIN ERROR unless EXP1 is a numeric integer 0 EXP1-INDEX (%%EXP2).

- RANK ERROR unless EXP1 is a scalar, and VAL is a scalar or else a vector of length %(EXP1 "do EXP2).
- DOMAIN ERROR unless VAL is of type integer.
- The type, rank, and number of the result are the type, rank, and number of EXP2.

43. exp tilde exp $

- DOMAIN ERROR unless EXP is of type bit.
- The type, rank, and number of the result are the type, rank, and number of EXP.

44. exp mfn exp $

- VALUE ERROR if EXP has a null value pointer.
- If the usage bead pointed to by the MFN token has been junked:
  - Deallocate the junked usage bead.
  - Search the symbol table for a new referent for the function name.
  - If not found, SYNTAX ERROR on the MFN token.
  - If found, but not a dyadic function anymore, SYNTAX ERROR on MFN.
  - Hook the MFN token to the new usage bead.
- Spawn an F-frame to the state stack, and hook it to the function's usage bead.
- Place the function argument in the symbol table with the proper value, and enter its usage pointer into the frame's local name list.
- If the function allows a result, enter the local name into the symbol table, and record its usage pointer in the frame's local name list.
- If the function has local variables, enter them into the symbol table, and record their usage pointers in the frame's local variable list.
- Scan the function's lexed lines for labels, enter them into the symbol table with their values, and record their usage pointers in the frame's local variable list.

- Go to the top of the parse main loop.
- When the function returns, place the result value pointer in the new EXP token.
- If the value pointer is null then set the PRINT switch OFF; otherwise, set it ON.

45. exp "ib exp $
- DOMAIN ERROR unless EXP is a numeric integer $19 \leq EXP \leq 27$.
- RANK ERROR unless EXP is a scalar.
- The result is a numeric integer scalar, except for EXP=27, which is a numeric integer vector.

46. exp sop exp $

47.1 exp ? exp $
- The rank and number of the result are the rank and number of EXP. The type of the result is as given in the chart.

47.2 exp % exp $
- The type of the result is integer, the rank of the result is 1, and the number of the result is the rank of EXP.

47.3 exp , exp $
- The type and number of the result are the type and number of EXP. The rank of the result is 1.

47.4 exp $ exp $
- DOMAIN ERROR unless EXP is a non-negative integer.
- RANK ERROR unless EXP is a scalar.
- The result is of type integer and rank 1. The number of the result is the value of EXP.

47.5 exp "tr exp $
- RANK ERROR unless %%EXP is 1, 2, or 3.

- The type, rank, and number of the result are the type, rank, and number of EXP.

47.6 exp "rf exp $

- The type, rank, and number of the result are the type, rank, and number of EXP.

48. exp gop exp $

- DOMAIN ERROR unless EXP is of numeric type.
- The rank and number of the result are the rank and number of EXP. The type of the result is integer.

49. exp "rr exp $

- The type, rank, and number of the result are the type, rank, and number of EXP.

50. exp sop / exp $

51. exp lop / exp $

- The type of f/E is the same of the type as EfE, in the chart. The rank of the result is one less than the rank of EXP. The number of the result is (#EXP)÷(%EXP)⌈%%EXP⌋.

52. exp sop / ⌈exp1⌋ exp2 $

53. exp lop / ⌈exp1⌋ exp2 $

- DOMAIN ERROR unless EXP1 is a numeric integer $0 \leq$ EXP1-INDEX $<$ (%%EXP2).
- RANK ERROR unless EXP1 is a scalar.
- The type of f/⌈E1⌋E2 is the same as the type of (E2)f(E2), in the chart. The rank of the result is one less than the rank of EXP2, and the number of the result is (#EXP2)÷(%EXP2)⌈EXP1⌋.

54. exp sop ⌿ exp $

55. exp lop ⌿ exp $

- The type of f⌿E is the same type as EfE, in the chart. The rank of the result is one less than

the rank of EXP. The number of the result is (#EXP)÷(%EXP) ⌈1⌋.

56. exp gop ⌈exp1⌋ exp2 $

- DOMAIN ERROR unless EXP1 is a numeric integer $0 \leq$ EXP1-INDEX $<$ (%%EXP2).
- RANK ERROR unless EXP1 is a scalar.
- DOMAIN ERROR unless EXP2 is numeric.
- The type of the result is integer. The rank and number of the result are the rank and number of EXP2.

57. exp "rr ⌈exp1⌋ exp2 $

- DOMAIN ERROR unless EXP1 is a numeric integer $0 \leq$ EXP1-INDEX $<$ (%%EXP2).
- RANK ERROR unless EXP1 is a scalar.
- The type, rank, and number of the result are the type, rank, and number of EXP2.

58. list exp $

- A list is created consisting of one value.
- Copy the "source" pointer from the EXP token into the LIST token in the event of a diagnostic later on.
- It is all right at this point for the expression to have a null value pointer.

Request Processing

Request processing is almost self-evident. A user-oriented description of what each request does is practically a flow-chart for it. The discussions below mention some points which might not be so obvious, or which involve interfaces with Multics.

)OFF

- The termination sequence for APL, as discussed in MSPM BZ.10.01, is executed.

)CONTINUE

- This request is mechanized as the sequence
)SAVE CONTINUE
)OFF

)QUIT

- APL calls the command processor through the entry point "cu_$cp". APL remains in the stack with the "cleanup" condition enabled. If the stack is released, APL will terminate as in the OFF request.

)CLEAR

- Any old copies of "apl.symbol.?", "apl.stack.?", "apl.function.?", and "apl.value.?" in the process directory are truncated.
- Empty stack and symbol table are instated.

)LOAD path

- The pathname given must be the pathname of a segment created with the APL SAVE request. Standard Multics search rules are applied if an incomplete path name is given.
- The current workspace is cleared.
- The saved workspace will consist of four areas. The four areas are simply moved into the working symbol, stack, function, and value segments. No relocation is necessary, as all lists are maintained as offsets relative to the beginnings of their respective areas.

)COPY)PCOPY)GROUP)ORIGIN)DIGITS
)WIDTH)WSID)FNS)VARS)GRPS
)GRP)SI)SIV

- The implementation of these requests is completely straightforward.

)SAVE path

- The four areas of the current workspace; namely, stack,

symbol, function, and value; are written one after the other as one segment. Standard Multics search rules are applied if an incomplete pathname is supplied. As discussed under LOAD, no relocation of any pointers is necessary to accomplish this. Only the portions of the areas actually used will be written out.

)DROP path

- Mechanized as the Multics DELETE command.

)LIB

- Mechanized as the Multics LIST command.

)PORTS

- Mechanized as the Multics WHO command.

)ERASE object...

- Groups and variables can always be erased. When a variable is erased, if its usage count in its usage bead is non-zero, the usage bead is set to type junk so that the user of the variable will be notified that it has been deleted.
- Functions cannot be erased if pendent. If a function is found with a non-zero usage count, the state stack will be searched to see if it is pendent. If it is pendent, the function wll not be erased. If it is not pendent, its usage bead will be set to junk, as discussed above.

)FO {path}

This request is processed differently depending upon whether it is encountered as a normal input line or as an argument to the APL command. As an argument to the APL command:

- If the normal output stream name is not "user_output", the name is detached and then set to "user_output_.
- If the given path name is not null, a unique identifier is placed in the normal output stream name, and then

this name is attached to the segment.

- The normal output stream name is placed in the current output stream name.

As a normal input line:

- If the diverted output stream name is not blank, it is detached and set to blank.
- If the given path name is null, thenormal output stream name replaces the current output stream name, and exit.
- Otherwise, a unique identifier is placed in the diverted output stream name, and the name is attached to the given segment.
- The diverted output stream name replaces the current output stream name.

)FI {path}

- The sequence of operations performed for the FI request are exactly the same as those for the FO request, above, with the word "input" inserted in place of "output".

)FIO

- This request is treated as a combined FI and FO request. No path name is permitted.

)FIA {path}

- The FIA switch is set and the pathname is remembered. When the parser receives the next complete line from the lexical analyzer, the following steps will be taken:
- The FIA flag is reset.
- If the diverted input stream name is not blank, it is detached and then made blank.
- If the saved pathname is null, the normal input stream name is placed in the current input stream name, and exit.
- Otherwise, a unique identifier is placed in the diverted input stream name, and the name is attached to the segment.
- The diverted input stream name is placed in the current stream name.

)FIOA

. This request is treated as a combined FIA and FO request. No path name is permitted.

)RO

This request is processed differently depending upon whether it is encountered as a normal input line or as an argument to the APL command. As an argument to the APL command:

. Exit immediately if the normal output stream name is "user_output".
. Detach the normal output stream name.
. Set both the current and the normal output stream names to "user_output".

As a normal input line:

. If the diverted output stream name is blank, replace the current output stream name with the normal output stream name and exit.
. If the current output stream name is "user_i/o", then replace it with the diverted output stream name and exit.
. Otherwise, detach the diverted output stream name and replace it with blank.
. Replace the current output stream name with the normal output stream name.

)RI

. Read "input" for "output" everywhere in the above description of the RO request.

)RIO

. The combination of an RI and an RO request.

)RIA

. The RIA switch is set. When the parser receives the next complete line from the lexical analyzer, the following steps will be taken:
. The RIA switch is reset.

- If the diverted input stream name is blank, replace the current input stream name with the normal input stream name and exit.
- If the current input stream name is "user_i/o", then it is replaced by the diverted input stream name and exit.
- Otherwise, the diverted input stream name is detached and replaced by blank.
- The current input stream name is replaced by the normal input stream name.

)RIOA

- The combination of an RIA and an RO request.