CGTM-11
J. Ehrman
December, 1966

SIMPLE-MINDED ELEMENTARY LIST LANGUAGE

A set of routines was recently written to enable users of higher-level languages (e.g. Fortran and Assembler) to perform list manipulations. These were written with no special use in mind; I would therefore be happy to receive comments and suggestions regarding additions, extensions, and corrections to the routines as described. Program decks are available for experimentation.

The basic structural unit is called a block, which contains two words of housekeeping information followed by 1 to 32 words of user-specified information which can be data or pointers to other blocks. The blocks are taken from a pool of 10000 words included in the routines; this is of course strictly an arbitrary choice, and the number and location of the blocks could be set up in some different way.

The routines to manipulate the contents of blocks are all called from a higher-level language. This was done so that a user can, for example, program a complicated operation on a previously created structure as a Fortran subroutine, which could then be called from some other program. In that sense the structure of the "language" is specified by the user who constructs the routines to call the ones provided by this set.

A. Use of Block-Structure Routines

1. To acquire a block

   P = GETBLK(n) $\quad 1 \leq n \leq 32$

   Causes a block pointer to a block of n elements (one fullword each) to be placed in P.

2. To print the contents of a block

   CALL DUMPBL(P) (where P is a block pointer)

   causes the contents of the block to be printed;

   CALL DUMPBL(n) $\quad 1 \leq n \leq 32$

   causes the contents of all blocks of size n to be printed;

   CALL DUMPBL(o)

   causes the contents of all blocks to be printed.

3. To Insert Items into a Block, and Obtain Items from a Block

   X = PUTDAT (P,I1,I2, ---,IN,D)

   causes the piece of data D to be placed in a block in a manner described below.

X = PUTPTR(P,I1,I2,---IN,PTR)
causes the block pointer PTR to be placed in a block in the following manner:

A. P is a block pointer to an initial block. The (I1)th word of that block is accessed.

B. If there is another indexing quantity I2, the previously accessed word is checked to see if it is a block pointer P1. If so, the (I2)th word of the block pointed to by P1 is accessed. If not, an error condition is recognized.

C. The next indexing quantity in the call is obtained, and another addressing cycle is performed, as described in step B above.

D. When the final addressing cycle is completed, the block pointer contained in PTR or the datum contained in D is obtained, and it replaces the word accessed in the final addressing cycle.

E. If the item being inserted is a block pointer, the reference count for the block to which it points is increased by one.

F. If the accessed word which is being replaced is a block pointer, the reference count of the block to which it points is reduced by one. If that reference count thereby becomes zero, that block is deleted from use and may be used for other purposes (e.g. other GETBLK calls), even though the block pointer obtained in the original GETBLK call may have been retained.

The quantity X indicated above has no particular value, and should not be used.

D = GETDAT(P,I1,I2,---,IN)
causes the datum referred to by the address string (which is calculated as in steps A-C above) to be returned as the value of the function GETDAT.

PTR = GETPTR(P,I1,I2,---IN)
causes the block pointer referred to by the address string to be returned as the value of the function GETPTR.

G. If the mode of the GETXXX call is different from the mode of the item finally accessed, an error condition is recognized.

H. If at any point in the addressing cycle described in B-C above an indexing quantity refers to an item not contained in the block being referenced an error condition is recognized.

B. Examples and Possible Extensions

At present the only checking done is to see if the implied mode of the routine being called corresponds to the mode of the argument (e.g. GETPTR must eventually get a block pointer, not a datum). Some further possibilities might be to have several levels of checking available which could be turned on and off dynamically; this would permit debugged routines to execute faster without error checks.

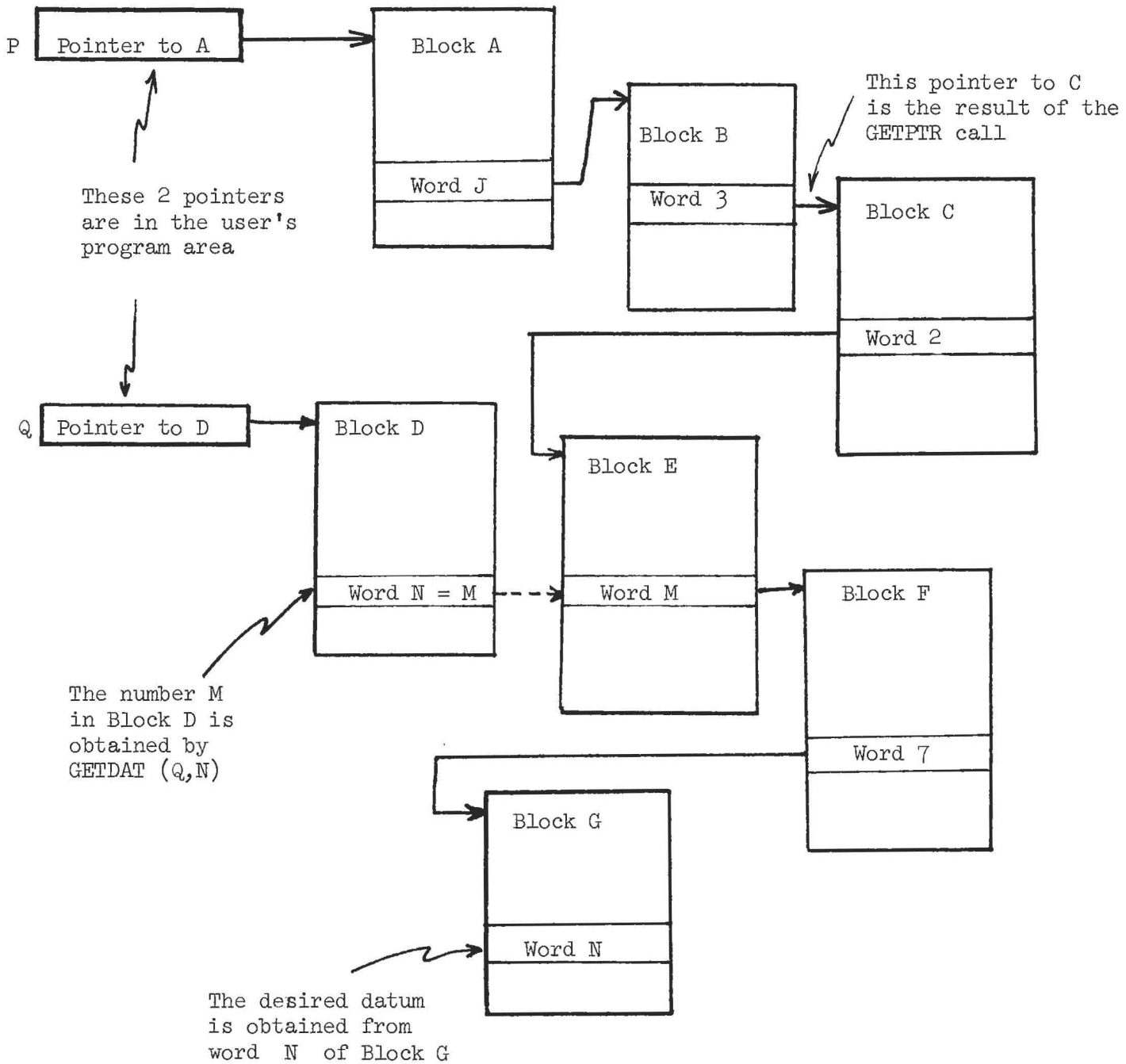
Garbage collection is handled automatically by the use of reference counts; it might be possible to improve the present routines by making the original pointer to the block unavailable to the user (for example by GETBLK returning an index in a list of pointers rather than the pointer itself).

One useful feature of these routines is that their use from a higher-level language implies some additional flexibility: for example, one could write

D=GETDAT (GETPTR(P,J,3),2,GETDAT(Q,N),7,N)

with such function calls nested as deep as desired. The above call is represented schematically by the following diagram.

Note that many other links may exist between the blocks, but only those used in this statement are shown.


P
Pointer to A
These 2 pointers are in the user's program area
Block A
Word J
Block B
Word 3
This pointer to C is the result of the GETPTR call
Block C
Word 2
Q
Pointer to D
Block D
Word N = M
Block E
Word M
Block F
Word 7
Block G
Word N
The number M in Block D is obtained by GETDAT (Q,N)
The desired datum is obtained from word N of Block G