

GVTDOC
D 211.
9:
4137

NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Maryland 20034



GRAPH INFORMATION RETRIEVAL LANGUAGE;
PROGRAMMING MANUAL FOR FORTRAN COMPLEMENT

LIBRARY

AUG 31 1973

by
S. Berkowitz, Ph.D.

U.S. NAVAL ACADEMY

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

COMPUTATION AND MATHEMATICS DEPARTMENT
RESEARCH AND DEVELOPMENT REPORT

20070119 189

June 1973

Report 4137

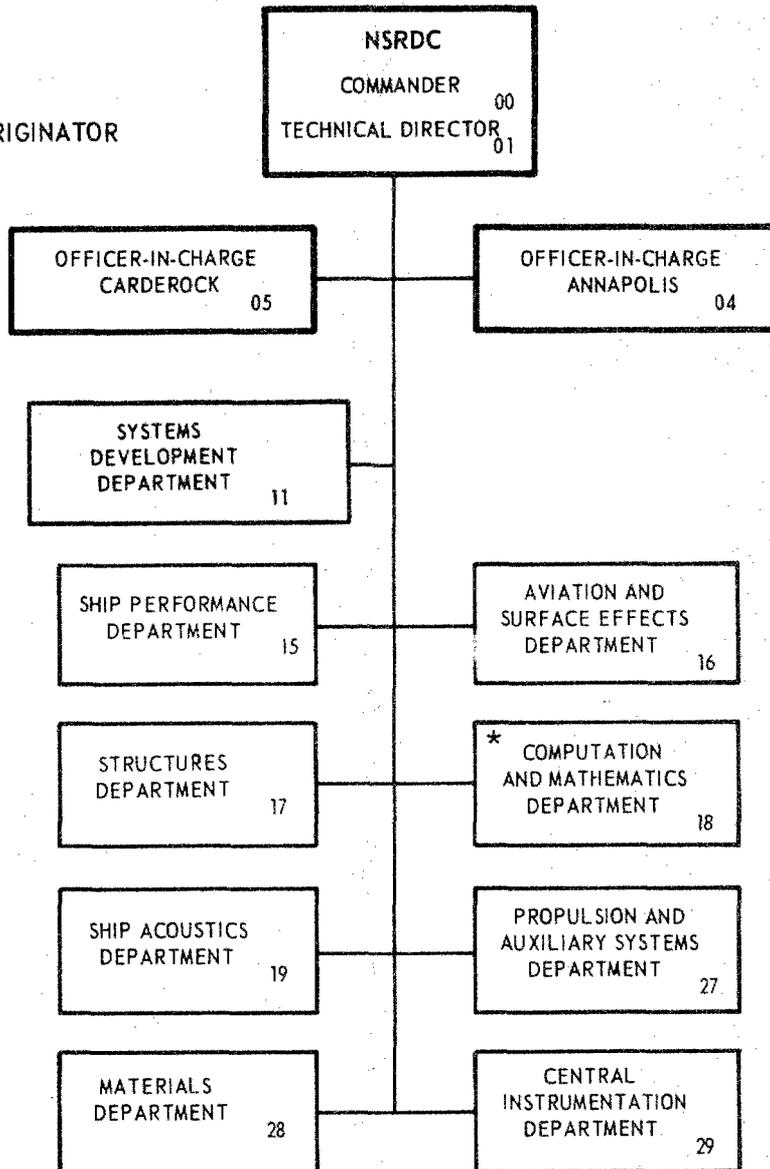
GRAPH INFORMATION RETRIEVAL LANGUAGE; PROGRAMMING MANUAL FOR FORTRAN COMPLEMENT

The Naval Ship Research and Development Center is a U. S. Navy center for laboratory effort directed at achieving improved sea and air vehicles. It was formed in March 1967 by merging the David Taylor Model Basin at Carderock, Maryland with the Marine Engineering Laboratory at Annapolis, Maryland.

Naval Ship Research and Development Center
Bethesda, Md. 20034

MAJOR NSRDC ORGANIZATIONAL COMPONENTS

*REPORT ORIGINATOR



DEPARTMENT OF THE NAVY
NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER
Bethesda, Maryland 20034

GRAPH INFORMATION RETRIEVAL LANGUAGE;
PROGRAMMING MANUAL FOR FORTRAN COMPLEMENT

by
S. Berkowitz, Ph.D.



APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

June 1973

Report 4137

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT.	1
ADMINISTRATIVE INFORMATION.	1
I. INTRODUCTION.	2
II. A GRADED PROGRAM OF EXAMPLES.	5
IDENTIFIERS, FUNCTIONS	6
INSERTION.	7
RETRIEVAL, INDEX	8
TRANSFER	9
DELETION	10
DATA, SEQUENTIAL SPACE	11
INCLUSION	15
RECOGNITION AND GENERATION	16
III. THE SYNTAX OF GIRL/FORTRAN.	18
THE GIRL/FORTRAN PROGRAM	18
BLANKS	19
DEFINITIONS, IDENTIFIERS	19
NUMERIC, HOLLERITH DATA.	20
TRANSFERS, LABELS, COMPARISONS	22
UNPARENTHESED STATEMENTS	24
1. Identification.	25
2. Insertion	25
3. Retrieval	23
4. Deletion.	28
5. Comparison.	28
6. Inclusion.	29
7. Indication.	29
PARENTHESED STATEMENTS	30
GIRL STATEMENT	32
IV. IMPLEMENTATION AND OPERATIONAL REQUIREMENTS	33
GIRL/FORTRAN TRANSLATION	33
MEMORY ALLOCATION.	33
GIRLS FUNCTION EXECUTION TIMES.	34

	<u>Page</u>
GIRL DECK SETUPS	36
Batch-Entry Deck Setups for a GIRL/FORTRAN Program.	37
Batch-Entry Deck Setup for Cataloguing a Graph Prior to Compression or Expansion	39
Batch-Entry Deck Setup for Graph Memory Compress or Expansion	39
NOTATION.	40
CHANGES IN THE MANUAL	41
ACKNOWLEDGMENTS.	42
APPENDIX A - EXPRESSIONS IN GIRL	43
APPENDIX B - GIRL MNEMONICS CARDS.	47
APPENDIX C - PARENTHESIZED STATEMENT BNF SYNTAX.	49

ABSTRACT

GIRL (Graph Information Retrieval Language) is a programming language designed to conveniently manipulate information in graph structures. As such, the language will play a key role in the construction of the organizational schemes found, for example, in information retrieval, pattern recognition problems, linguistic analysis, and process scheduling systems. The language is written to complement an algebraic language, in the sense that GIRL statements are distinguished from the statements of the algebraic language and the statements may be interleaved. The primary advantage of separating symbolic and numeric statements is that the programmer is afforded a linear, one-one trace of graph operations in the code description.

ADMINISTRATIVE INFORMATION

The work of this report was carried out in the Computer Sciences Division under the sponsorship of SHIPS 00311, Task Area SR0140301, Work Unit 1-1834-001..

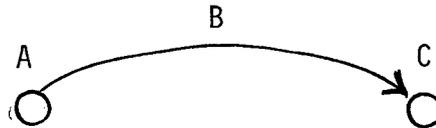
I. INTRODUCTION

GIRL (Graph Information Retrieval Language) is a programming language designed to conveniently manipulate information in graph structures. As such, the language will play a key role in the construction of the organizational schemes found, for example, in information retrieval, pattern recognition problems, linguistic analysis, and process scheduling systems. The language is written to complement an algebraic language, in the sense that GIRL statements are distinguished from the statements of the algebraic language and the statements may be interleaved. The primary advantage of separating symbolic and numeric statements is that the programmer is afforded a linear, one-one trace of graph operations in the code description. From an opposing point of view, Feldman and Rovner's LEAP¹ and Ross' AED-0², for example, are extensions of ALGOL in the sense that graph or list operations are interspersed with numeric operations. The result is that code sequencing of graph operations is bound by the infix, phrase substitution nature of the algebraic language, and does not lend itself to an easy scan of the graph. On the other hand, the ALGOL extensions offer a uniformity of notation necessarily missing from GIRL.

¹ Feldman, J.A. and Rovner, P.D., "An ALGOL-based Associative Language," Communications of the Association for Computing Machinery, Vol. 12, No. 8, pp. 439-449 (1969).

² Ross, D.T., "A Generalized Technique for Symbol Manipulation and Numerical Computation," Communications of the Association for Computing Machinery, Vol. 4, No. 3, pp. 147-150 (1961).

Graphs are composed of structures of (source node)-(link)-(sink node) triples, one of which is illustrated here. One may think of such



a structure as a function B of argument A, and value C, read as: "B of A is C". Moreover the function can be multivalued in which case B points to an ordered set of sink nodes, a list.

The function of GIRL is to insert, identify, retrieve, delete, and compare node-link-node triples. One purpose of GIRL is to serve as a base for a more sophisticated language called PIRL³ which will generalize the GIRL range of arguments from nodes and lists to arbitrary, directed graph structures. Nonetheless, GIRL is a powerful and efficient language in its own right, comparable in scope say to LISP⁴. Whereas LISP is based on a recursive function structure—in theory at least—GIRL acknowledges the need for indexed iteration and labelled transfers, while at the same time permitting recursive functions if they are permitted by the complementary algebraic language. In addition, the arithmetic capability of GIRL/FORTRAN or GIRL/ALGOL is as good as that of the algebraic language, whereas the arithmetic capability of LISP has been traditionally awkward at best.

³ Berkowitz, S., "PIRL - Pattern Information Retrieval Language - Design of Syntax," Proceedings of 1971 National Conference of the Association for Computing Machinery, pp.496-507.

⁴ McCarthy, J., et al., "LISP 1.5 Programmers Manual", MIT Press, Cambridge, Massachusetts (1962).

Perhaps the best way to learn GIRL is by studying and working out examples. Accordingly, the next Chapter presents a graded series of example graph problems, some worked out, some not. The alternative approach would be to study the syntax and to work the examples by deduction. The deductive approach is not recommended since the language can provide very complex code. The inductive approach, on the other hand, begins with simple linguistic structures which lend themselves well to intuitive generalization.

Chapter III presents the detailed syntax of GIRL. The symbols used specify the implementation language. The format of presentation is as follows: a fragment of syntax in Backus-Naur form (BNF) or a modification thereof, followed by explanatory semantics, and concluded with annotated examples or a reference to the examples in Chapter II.

The last Chapter contains details of implementation, such as publication notation, memory allocation, control cards, run-times. Since the language or its implementation may be expanded or revised, updates to the Manual will appear from time to time.

The rationale for the form of the GIRL language may be found in the original design paper⁵. Since writing the paper, moreover, other considerations have shed more light on why a graph processing language should have the form of GIRL. Some of these considerations are reviewed in the Manual and some will be issued as future attachments.

A version of GIRL has been implemented and operational since 1969. It has been used in syntactic parsing, pattern recognition, sparse matrix computation, information retrieval, network design, and an auditing compiler.

⁵ Berkowitz, S., "Graph Information Retrieval Language - Design of Syntax", in "Software Engineering" edited by J. Tou, Vol.2, Academic Press, New York, pp. 119-139 (1971).

II. A GRADED PROGRAM OF EXAMPLES

This chapter presents a series of examples designed to give the reader a notion of the power and flexibility of GIRL. The examples are graded within each section in the sense that they become progressively more complex, and they are cumulative in the sense that an example may require knowledge gleaned from previous examples. However, the examples do not comprehend all the details and structural variations of GIRL/FORTRAN. Rather, if the reader carefully studies each example to see what it presents that he has not seen before, he is sure to ask a question beginning "What if...?" It is precisely this spark of curiosity that we intend the Chapter to enkindle. A reader with a question of this software will find the answer by a study of the appropriate semantics and syntax (in that order) displayed in Chapter III.

In the examples, a personnel file is constructed and manipulated. The reader should understand that the example system is neither a complete nor even a suitable way to approach the problem area. Rather, the example system offers complex data structures that we use for convenience in teaching GIRL. An excellent way for the reader to test his knowledge and to appreciate the conciseness and legibility of GIRL would be to code more realistic programs in areas such as information retrieval, job shop simulation, program management, or syntactic analysis.

For the sake of reference, a table of GIRL expressions is supplied in Appendix A. A mnemonics cut-out card is provided in Appendix B.

IDENTIFIERS, FUNCTIONS

1) Create a random node and call it NOUN2:

```
G  $'NOUN2
```

2) Create the random nodes NOUN1, NOUN2, NOUN3:

```
G  DEFINE NOUN1,NOUN2,NOUN3
```

3) Create a random node and call it both X1 and X2:

```
G  $'X1'X2  
or G  $(X1,X2)  
or G  $('X1,'X2)
```

The identify (sometimes called name, define) operation ' offers a preview of some basic principles of PIRL/GIRL, namely:

- Statements are read and executed strictly from left-to-right, leaving a value after the execution of each operation. The value effectively initiates a new left-to-right execution (or statement termination). In the case of identification, the operation leaves the previous value unchanged.
- A statement may be broken at any point to serve as a prefix to a parenthesized list of suffixes. Since execution occurs strictly from left-to-right, the value or operation remaining in front of the left parenthesis serves as the initiator for all the suffixes. Since nesting may occur to any level, a GIRL statement is a right-branching tree.
- Identifiers are FORTRAN integer variables whose value is the address of some node.

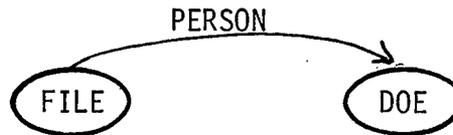
4) If RAND(S) is a FORTRAN function which generates an integer from some algorithm (say a uniform density on [0, MEMSIZE] where MEMSIZE is the graph memory size), then name the resulting number, i.e., node, X1 and multiply it by 2:

```
G  *RAND(S)'X1  
X1=X1*2
```

Unless explicitly stated otherwise, the identifiers used in the following examples are all defined.

INSERTION

5) Insert the following graph:



G FILE PERSON DOE

6) Add SMITH and JONES to the file graph in (5)

G FILE PERSON (SMITH, JONES)

7) Describe SMITH by linking him with the attribute MARRIED to a new node to be called NO, with the attribute SUPPORT to himself, and with the attribute SUPERVISES to JONES, DOE, and BROWN:

G SMITH(MARRIED \$'NO,SUPPORT SMITH,SUPERVISES (JONES,DOE,BROWN))

8) Similarly, JONES is not married, supports himself, DOE, and two others, works in PROJECT1 under SMITH:

G JONES(SUPPORT(JONES,DOE,"2"),MARRIED NO, WORKSIN PROJECT1 UNDER SMITH)

(Note the integer data insertion "2".)

9) What does the file graph look like now? How many multivalued lists are there? How many circuits? (e.g., JONES WORKSIN PROJECT1 UNDER SMITH SUPERVISES JONES is a circuit.)

RETRIEVAL, INDEX

10) Retrieve the first and second values of the JONES SUPPORT link, and call the values V1 and V2, respectively. Find the person V2 SUPPORTs and call him V3

```
G JONES+SUPPORT('V1, .2'V2+SUPPORT'V3)
```

That is, A+B elicits the first value on its value list. Lists are not named in GIRL! (A+B.1 is the same as A+B in GIRL.)

11) Test whether or not the file graph contains information about DOE's marital status. If it does, go to statement 300, otherwise to 400.

```
G DOE+MARRIED/400/300
```

12) Find the second person whom JONES SUPPORTs and ask whether it is DOE. If not, go to NEXT; if so, continue to find whom JONES SUPERVISES and whom the first such person SUPPORTs. Call the latter V4.

```
JONES+(SUPPORT.2=DOE/NEXT, SUPERVISES+SUPPORT'V4)
```

Note that a point (.) indicates an operation, just as does the plus (+) sign. That is, .2 applies not as index to SUPPORT, but to the result of the preceding prefix phrase JONES+SUPPORT. Similarly, SUPERVISES+SUPPORT is not a phrase; rather, SUPERVISES applies to JONES+ and then (in strict left-to-right scan) +SUPPORT applies to the result of JONES+ SUPERVISES.

Note also that /NEXT is a failure transfer, with "continue" on success. Similarly, //NEXT would be a success transfer with "continue" on failure. All operations may be tested for success or failure.

TRANSFER

13) Is the third person on the file graph married? If so, go to statement 300; if not, go to statement 400.

G FILE+PERSON.3+MARRIED=NO/300/400

Be careful. Compare the transfer here with the previous example. The word NO has no particular meaning in the graph. The transfer tests only the success or failure of the previous operation.

14) Name all the multivalued lists upon which the node JONES can be found. Will FILE+PERSON.I=BROWN succeed for some value of I?

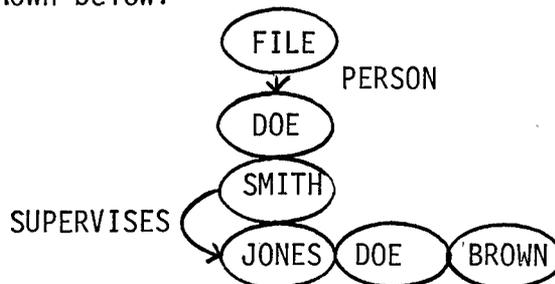
JONES can be found on:

- the PERSON list from FILE along with DOE and SMITH
- the SUPERVISES list from SMITH along with DOE and BROWN
- the SUPPORTS list from JONES along with "I2"

Though the single node JONES can be found on each of these three lists, each list retains its individual structure as illustrated in the following diagrams, so that nodes on one list cannot be accessed by indexing another list.



That is, although JONES is on both the lists diagrammed, one cannot access BROWN by writing FILE+PERSON.5 as one might think if he were to draw the picture as shown below:



15) Let us give another name to the project that JONES WORKSIN: we will call it WORKSIN. Can we still find the project that JONES works in? If yes, go to 350; otherwise, continue to the next statement.

```
G JONES(+WORKSIN 'WORKSIN, +WORKSIN//350)
```

The interpretation of this statement is a matter of definition. In order to illustrate the definition, we replace each identifier with its defined value in a strict left-to-right scan.

Suppose the initial values of JONES, WORKSIN, PROJECT1 are 100, 300, 500, respectively. That is, GIRL identifiers are FORTRAN variables, and have values which might, by the way, be detrimentally changed in the complementary FORTRAN program if the programmer were careless. Then the statement has the evaluation:

```
JONES+WORKIN: 100+300 has the value 500
'WORKSIN: WORKSIN has the new value 500
JONES+WORKSIN: 100+500 has no value
```

```
∴ The last retrieval operation fails and control passes to
the next statement
```

DELETION

16) Correct the damage done to the file graph in 15, delete the MARRIED link of SMITH, verify that he is not DOE's supervisor. If in fact he is DOE's supervisor, go to 350; otherwise continue to ask whom he does supervise and label the first such person as EE.

```
WORKSIN=300
I=0
400 I=I+1
G SMITH(- MARRIED, +SUPERVISES.I/450#DOE/350/400)
G 450 SMITH+SUPERVISES'EE
350
```

An equivalent code is

```
        WORKSIN=300
        I=0
G  400  SMITH(-MARRIED, +SUPERVISES."I=I+I"/450#DOE/350/400)
G  450  SMITH+SUPERVISES'EE
        350
```

That is, double quotes may enclose an arbitrary, integer-valued FORTRAN arithmetic statement—or arithmetic expression for that matter—and thus, in this case, embody the iteration within the GIRL statement.

DATA, SEQUENTIAL SPACE

17) DOE has BS, MS, PHD degrees (Hollerith data), and received them in years 1956, 1958, and 1961 (integer data), respectively. Add this information to DOE's dossier by interleaving degree and data as Hollerith and integer data, respectively.

```
G  DOE DEGREE('//BS',"1956", '//MS',"1958", '//PHD',"1961")
```

18) For quicker access, let us repeat the previous example, but rather put the data into sequential space; i.e., into the array SEQ, beginning; say at SEQ(31). We might also modify the BS and PHD degrees by the attribute GRANT, indicating, say, that DOE attained the degree under a company grant. Note now that the cells in each block of SEQ must be of a uniform data type.

```
G  DOE(DEGREE;31;3 '/30/BSbbbbbbbbMSbbbbbbbbPHDbbbbbbb',
        GRANT(;35;3 YES, .2 NO, .3 YES),
        WHEN(;39;3 "1956", .2 "1958", .3 "1961"))
```

The indices refer to item locations on the respective lists. It would have been more efficient to have placed all items except the first of each list into SEQ by direct FORTRAN statements.

If one were to look at the vector SEQ, one would find:

<u>Address</u>	<u>Contents</u>	<u>Form</u>
SEQ(31):	3	(number of items in block)
(32):	BS	(Hollerith)
(33):	MS	(Hollerith)
(34):	PHD	(Hollerith)
(35):	3	(number of items in block)
(36):	YES	(identifier)
(37):	NO	(identifier)
(38):	YES	(identifier)
(39):	3	(number of items in block)
(40):	1956	(integer)
(41):	1958	(integer)
(42):	1961	(integer)

Note that although SEQ affords quicker indexing access, it also constrains one to specify and remember the dimension of each block of SEQ beforehand, thus losing the dynamic storage capability of nonsequential memory. On the other hand, note that any node identifier placed in sequential memory may be linked further just as in nonsequential memory. Thus,

DOE+GRANT+WHEN/FALL

is perfectly legal even though in our case it would FAIL since neither YES nor NO has yet been given a WHEN LINK.

19) Find DOE's second degree, call it DEG, and change its associated date to 1957 (not for the SEQ interpretation).

G DOE(+DEGREE./2'DEG, DEGREE .,2 "1957")

Do the same for the SEQ interpretation.

There are index retrieve, insert, and delete operations at a given list location both for data of unspecified type and also for specified identifier, number, and Hollerith data. The operations are multi-symbol and have the following mnemonics: The point (.) indicates an index.

The minus (-) preceding an index point indicates a deletion at the indexed location. (When the deletion follows the link identifier of an insertion, the insertion is destructive at the designated location.) When a minus (-) follows an index point, it indicates a list location to be found by counting up from the bottom of the list. A slash (/) indicates Hollerith data, another point (.) indicates numerical data, and an equal sign (=) indicates an identifier. Thus, `-.3` means "delete third numerical item on the preceding list." Similarly, `-.3` means "delete the third item on the preceding list." Again, coding `-.!"I=I+1"` would mean "retrieve the Ith numerical item counting from the bottom of the list, where I has been replaced by I+1." The table at the end of the chapter lists the possible combinations. Note that identifier values (integers) and numeric integer data are distinguished, even in sequential space.

20) Give at least three expressions that will change DOE's PHD to an MD. (There are more than three.) One might use the facts that `./M` will access (by index) or insert the Mth Hollerith datum, that `-./M` eliminates such a datum, that `.M` accesses by index the Mth item closes up the list and leaves as its value the M+1th item.

21) Let us look a bit more closely now into the semantics of retrieval, insertion, deletion, and indication.

G DOE+DEGREE

retrieves the first element of a list. But

G DOE+DEGREE.3

increments the retrieval to indicate the third element of the list. Now

G DOE-DEGREE

deletes the whole list (and leaves DOE +DEGREE)!! Why not delete just the first element, to be consistent? Or on the other hand, why should DOE+DEGREE

not retrieve the whole list? Indeed, the latter question is the key to our rationale. We do wish to retrieve the whole list, but in GIRL we have no means of identifying or stacking the list. In PIRL³ this will be remedied. How, then, shall we now delete DOE's third DEGREE?

```
G DOE+DEGREE-. /3
```

How shall we insert MA as the second degree, while at the same time eliminating the previous MS degree?

```
G DOE(+DEGREE-. /2, DEGREE . /2 '//MA')
```

or, more concisely, we imbed the retrieval tacitly in an insertion:

```
G DOE DEGREE -. /2 '//MA'
```

22) Push the DEGREE list down by adding another BS at the top:

```
G DOE DEGREE . /1 '//BS'
```

Pop the DEGREE list back.

```
G DOE+DEGREE-. /1'DEG
```

What is the value of DEG? It is DOE+DEGREE. /1 (before deletion).

23) Now replace the whole DEGREE list by an MA.

```
G DOE DEGREE - '//MA'
```

That is, minus (-) without index after a link identifier represents a destructive insertion.

24) Do the following statements mean anything?

```
G DOE-DEGREE.2  
G DOE -.2 DEGREE
```

No, since .2 is an operational on a list, and as yet the list has not been accessed. (In PIRL, however, these will be legal since a list may have been identified by DOE.)

25) JONES' monthly expense ACCOUNT is located in a matrix ACNT. Point to it.

```
G JONES ACCOUNT "LOC(ACNT)"
```

26) Now retrieve and update the I^{th} item in JONES' ACCOUNT by \$3000, presuming ACNT to be in labelled COMMON. Note that LOC is a CDC FORTRAN EXTENDED function to compute the address of its argument.

```
G JONES+ACCOUNT 'J
  K = J-LOC(J)+I
  J(K) = J(K)+3000
```

The matrix ACNT has to be in labelled COMMON in order to preserve its relative address from execution to execution.

27) Conversely, if JONES' account had originally been put into ACNT, the ACNT could be inserted into the graph by

```
DO 1 I=1,N
G 1 JONES ACCOUNT "ACNT(I)"
```

What if his ACNT held a list of the types of expenses (i.e., identifiers)?

INCLUSION

28) Find out if SMITH is in the PERSONnel FILE.

```
J=0
G AGAIN FILE+PERSON."J=J+1"/NO=SMITH/AGAIN
```

or, equivalently,

```
G FILE+PERSON :SMITH/NO
```

29) The inclusion operation : is read as "contains". Which FILE location contains SMITH?

```
G FILE+PERSON :SMITH/NO'LOC
```

30) Eliminate SMITH from the FILE.

```
G FILE+PERSON(:SMITH/NO 'LOC, -.LOC)
```

or, equivalently, since the value of :SMITH is its list location,

```
G FILE+PERSON -.:SMITH
```

31) Put SMITH back into the FILE.

```
G FILE PERSON .2 SMITH
```

32) Put SMITH back at the end of the FILE only if he is not already in the FILE.

```
G FILE(+PERSON :SMITH//IN, PERSON SMITH)
```

or, in abbreviated form,

```
G FILE PERSON :SMITH//IN SMITH
```

33) Replace SMITH by BROWN only if JONES is in the FILE.

```
G FILE PERSON :JONES/NO .:SMITH BROWN
```

That is, add to the FILE of PERSONnel which contains JONES the replacement of SMITH by the name of BROWN. This is shorthand for

```
G FILE(+PERSON(:JONES/NO, #SMITH/NO 'LOC),PERSON .LOC BROWN)
```

RECOGNITION AND GENERATION

34) Note that the PERSONnel FILE is basically a generative list. When a recognition query is posed, the membership operation is necessary, but if such queries were common, it might be easier to restructure the FILE as a recognitive tree. For example, one might insert:

```
G FILE(DOE FILE, SMITH FILE, JONES FILE)
```

Then asking for SMITH or deleting him would have the respective forms:

```
G  FILE+SMITH
G  FILE-SMITH
```

The foregoing distinction between cognitive and generative memory structures is fundamental. The inclusion operation offers an abbreviated notation for accessing a list by content rather than by index, and thus offers an effective, temporary restructuring of a generative list for recognition purposes. Since indication seems to be the complement of inclusion, one might think that a restructuring of a cognitive relationship should be available, namely: xpJ meaning generate the J^{th} link of the node x .

This reasoning is correct, but the underlying hashed-address memory scheme does not yet permit such commands to be realized. The situation will be remedied in the PIRL language³, in which subgraphs are handled much as generalized nodes in the sense that one may query memory as to instances of a subgraph schema of given form but only partially specified node or link component. Due to the structure of the currently simulated associative memory, if one wanted the link list of a node available, the only practical way to handle it would be to explicitly imbed the list in the graph via a distinguished link.

III. THE SYNTAX OF GIRL/FORTRAN

THE GIRL/FORTRAN PROGRAM

A GIRL/FORTRAN source program consists of interleaved blocks of GIRL and FORTRAN statements. The form of FORTRAN statements is specified elsewhere⁶.

GIRL statements are distinguished by a G in the first column. In all other respects, the statement identification, continuation, statement, and comment fields for punched-card/tape format are exactly the same as for FORTRAN.

The labels which fill FORTRAN statement identification fields are restricted to positive integers less than 20000. GIRL labels are discussed below.

In the syntax, the metasyntactic symbols ::= (is defined to be) and | (exclusive or) are used as part of the usual Backus-Naur (BNF) notation. A string of small Roman characters represents a syntactic category and FORTRAN Hollerith characters form the terminal alphabet. Phrases enclosed between double lines ||...|| are descriptions of categories easily understood but not easily represented by a usual syntactic description. For example,

emp ::= ||empty category|| (0)

represents the empty category; i.e., the null alternative when "emp" appears in the right-hand side of a syntax statement.

⁶ "American Standard FORTRAN"; American Standards Association, New York (1966).

BLANKS

◦ Semantics:

Blanks are required only to separate identifiers and/or labels not otherwise separated by operators or delimiters. Otherwise, blanks may be used without restriction. The syntax will not formally specify the use of blanks since they are properly a lexical concern.

DEFINITIONS, IDENTIFIERS

◦ Syntax:

<u>define</u>	::= DEFINE idf1	(1)
idf1	::= idf idf1, idf1	(2)
idf	::= FORTRAN alphanumeric identifier	(3)
<u>id</u>	::= idsub i	(4)
i	::= 'idf i 'idf emp	(5)
idsub	::= idf \$ * idf (idcf1)	(6)
idcf1	::= idf cons idf (idcf1) idcf1, idcf1	(7)
cons	::= FORTRAN constant	(8)

◦ Semantics:

Identifiers must be defined before use as FORTRAN variables by giving them the value of an internal node of the graph. One way to do this is to use a DEFINE statement (define), especially when a large number of variables are to be assigned node addresses at once. The DEFINE statement must be the first executable statement in a GIRL/FORTRAN program, and there is at most one such statement in the program. The identifier list (idf1) which forms the argument of DEFINE is of arbitrary length. Although the defined identifiers are in fact integer variables, the programmer should not explicitly declare them to be so. A means of definition by the operator quote (') is given in Equation (5). Any variables thus defined must be tacitly INTEGER or explicitly declared to be so.

FORTRAN alphanumeric identifiers (idf) and FORTRAN constants (cons) are defined elsewhere.⁶ However, a restriction is that identifiers may not begin with LV. By writing the symbol \$, one can generate a random internal address, not otherwise used, in place of an identifier. Similarly, a uni-valued function (* idf (idcf1)) may be defined exactly as a (non-recursive) FORTRAN function subprogram⁶ and used in place of an identifier.

◦ Example:

- (i) Three ways to have X and Y reference the same randomly generated node:

```
*RANDM(START) 'X 'Y      , where RANDM generates a
                           random number between 1 and
                           the graph memory size.

$ 'X 'Y
DEFINE X, Y
```

- (ii) See examples 1 through 4 in the previous Chapter.

NUMERIC, HOLLERITH DATA

◦ Syntax:

```
data      ::= "aes" | '//hnq' | '/integer/ h'      (9)
aes       ::= ||FORTRAN arithmetic expression or statement|| (10)
hnq       ::= anq | h anq                          (11)
h         ::= a | h a                              (12)
anq       ::= digit | A,...,Z | + | - | . | | * | ( | ) | = | # | ,
           | ; | " | ' | :                          (13)
a         ::= anq | '                              (14)
integer   ::= digit | integer digit                (15)
digit     ::= 0,...,9                              (16)
dseq      ::= ;ivc; ivc | emp                       (17)
ivc       ::= idf | int | "aes"                    (18)
```

◦ Semantics:

In Equation (13), the notation A,...,Z stands for a disjunction of the letters of the alphabet; similarly, for 0,...,9 in Equation (16).

Sink nodes, besides representing identifiers, also represent graph-held or matrix-held data (data) expressed as an arithmetic statement or expression (aes), as an uncounted Hollerith string (hnq) without quote ('), or as a pre-counted Hollerith string (h) possibly including a quote. The length of the pre-counted string must match the integer between slashes, whereas the length of the uncounted string is bounded only by the capacity of memory. Hollerith strings of ten characters or less may be stored in the graph or in the matrix SEQ. A longer Hollerith string must be stored in one block of the matrix SEQ, Each SEQ matrix cell holds ten Hollerith characters except for the last cell of a block which contains ten or less characters, left-justified. Both identifiers and numeric data occur as sink nodes either in the graph or in SEQ. In order to set up a data block in SEQ, one uses the declaration dseq (see Equation (17)) in an insertion operation, as discussed later for Equations (33) and (34). In Equation (17) the first integer variable or constant (ivc) represents the address of the head cell of a SEQ block. The second ivc whose value is stored in the head block cell, represents the number of items in the block. The data of any block follows the head cell of the block and its type is homogeneous throughout the block. The type of data is denoted by the indicator "ivc" for numeric data and the indicator '/' for Hollerith data as shown in Equation (9). These indicators initiate the data declared by dseq as shown later in Equation (34). The absence of such indicators signals a block of node identifiers. Clearly the speed with which one can manipulate block data must be balanced in use against the rigidity of a block structure.

◦ Example:

Review example (18) in the preceding Chapter.

TRANSFERS, LABELS, COMPARISONS

◦ Syntax:

ti ::= i | i ti | /labeli | //labeli | /labeli/labeli | eqs
| eqs ti | emp (19)

labeli ::= i label i | label i (20)

label ::= idf | integer (21)

eqs ::= eqs eqs | eqn divc ti | emp (22)

eqn ::= = | # (23)

divc ::= data | ivc (24)

◦ Semantics:

A label (label) may take one of three forms:

- (a) relative GIRL address—an identifier (id) of less than six characters not beginning with V. Relative addresses do not refer to FORTRAN statements.
- (b) absolute GIRL or FORTRAN address—a positive integer less than 20000..
- (c) variable GIRL or FORTRAN address (for use as the argument address of a transfer operation only)—a FORTRAN integer variable beginning with V, having an absolute address as value.

A label either labels a GIRL/FORTRAN statement by residing in the statement identification field⁶ or points to a statement for control purposes as the argument of a transfer operation.

The two argument addresses of a transfer (ti) indicate the statement address to which control is transferred upon failure or success, in that order, of the operation immediately preceding the transfer. If the first

(or second) argument address is empty, the occurrence of a failure (or success) condition indicates that control is not altered; that is, the remainder of the statement is to be executed. Also, control is not altered for success or failure if no transfer whatsoever is specified.

Finally note that DO loops may not terminate on a GIRL statement. Rather, they must terminate on a FORTRAN statement (e.g., a CONTINUE statement).

Identification (i) is explained in a previous section and comparison (eqs) is described in the next section. The function of identification is to name the value that precedes it. The function of comparison is to test the value that precedes it for equality with what follows. These operations are included here for syntactic convenience since their value is the same as the value preceding the operation.

◦ Examples:

- (i) Give the name NEW to the preceding identifier and transfer to the relative address REL if the preceding operation failed, or to the absolute address 250 if the operation succeeded:

X'NEW/REL/250

- (ii) If the preceding operation succeeded, name the resulting value NEW and transfer to the variable address VAR, otherwise continue:

//'NEW VAR

- (iii) Review examples 7, 8, 9 in the preceding Chapter.

UNPARENTHESIZED STATEMENTS

◦ Syntax:

unpar ::= id usuff (25)

usuff ::= suff | dsuff (26)

suff ::= id ti pm ix ti dseq id ti
 | + id ti pm ixn ti
 | bo id ti
 | 'idf ti
 | suff suff (27)

dsuff ::= id ti pm ix ti dseq data ti
 | + id ti : obj ti
 | suff dsuff (28)

bo ::= + | - | = | # (29)

ixn ::= ixo ivc
 | ixo : obj (30)

ix ::= ixn | emp (31)

obj ::= ivc | data (32)

ixo ::= . pm ixtyp
 | . ixtyp pm (33)

pmn ::= + | - (34)

pm ::= pmn | emp (35)

ixtypn ::= ' | / | . (36)

ixtyp ::= ixtypn | emp (37)

◦ Semantics:

GIRL unparenthesized statements (unpar) are evaluated on a strict left-to-right scan. The evaluation of an operation may be either an identifier node or a data node. An identifier node may serve as prefix to a suffix string (suff), but a data node is an evaluation instance of a data suffix (dsuff) and is terminal. The basic operations are now explained.

1. Identification

Although identification has been discussed previously, we now comment on its use in the context of an operation sequence. Quite simply, the operation suffix 'idf gives a name to the last evaluated node or link. The operation always succeeds and is transparent to a transfer of control. That is to say, a success test following an identification refers to the last non-identification operation.

2. Insertion

A node-link-node triplet is inserted by juxtaposition of the respective identifiers. For example, A B C. Insertion is non-destructive, so that if A B C has been inserted, then inserting A B D will add D to the list linked to A by B. Furthermore, since a multivalued function generates its value set in the order of insertion—i.e., generates a value list—the repetition of an insertion triple induces a repetition of sink nodes on the value list. The presence or absence of at least one value on a value list before insertion may be tested for either directly after the link identifier—by the first ti of the first alternative of suff or dsuff—or after the insertion—by the last ti of the first alternative of suff or dsuff. For example, A B /X C transfers control to X if the value list is empty, whereas A B C /X inserts A B C and then transfers control to X if the value list had been empty. The value of an insertion is the identifier or data inserted.

By placing an index (ix) after the link identifier, one can insert destructively or non-destructively at a specific location in the value list, counting from the top or bottom. The success of the index retrieval may be tested (ix ti of suff or dsuff). The formalism of the index ix will be detailed in the description of the indication operation.

A sink node may be placed in the matrix SEQ by using the dseq mechanism (cf. Equation (17)). Each time dseq is used, a new block is declared, and it is the programmer's responsibility to avoid block conflicts. The dseq declaration stipulates two numbers. The first number indicates the location in SEQ of the head block cell. The second number, which resides in the head block cell, indicates the location in SEQ of the last block cell. After the first sink node has been placed in a SEQ block by dseq declaration, other nodes of the same type may be put in the same block either by a GIRL indexed insertion or by direct FORTRAN store. SEQ storage and graph storage may be mixed on the same list and the search for a list cell of given index will traverse SEQ block cells. However, a Hollerith string of more than ten characters must be stored only in SEQ and a Hollerith SEQ block is considered to be a single cell as far as an index operation is concerned. An attempt to insert more than ten Hollerith characters without a dseq declaration will result in the first ten characters being placed in the non-SEQ associative store as a single Hollerith cell. The purpose of SEQ is to allow rapid GIRL retrieval of list information for lists of fixed length in such a way that the GIRL retrieval mechanism is blind to the storage medium, be it graph or SEQ. (However, for a Hollerith retrieval, as will be discussed in the next section, a special variable is required to distinguish between SEQ and non-SEQ storage.) The underlying associative memory operation is quite sensitive to excursions into SEQ, it is usually not a good idea to mix short SEQ blocks with graph nodes; nor is it wise to insert or delete in SEQ by means of GIRL—FORTRAN should be used instead. A detailed example is in order.

◦ Example of Insertion

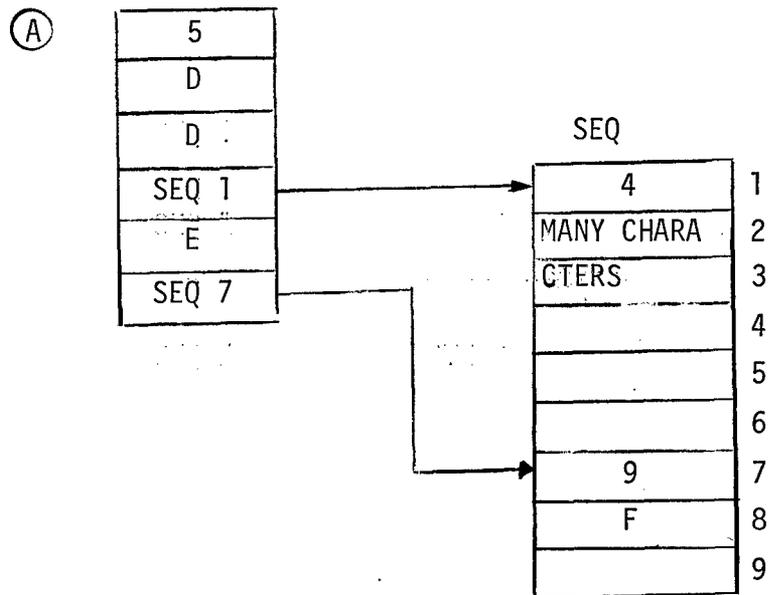
Consider the following program of insertions:

```

X = 3
G A B "X=X+2"
G A B D
G A B D
G A B ;1;4 '//MANY CHARACTERS'
G A B .5 E
G A B ;7;9 F

```

The resultant graph has the structure:



At this point, A B . 7 "3", which attempts to put '3' into SEQ(9), would fail because of a type error. That is SEQ(7) heads an identifier block. On the other hand, A B .6 G would succeed—albeit inefficiently—in putting G in SEQ(8) and moving F to SEQ (9).

Finally, we note a textual variant; namely, X Y - Z is the same as X(-Y, Y Z). (The notation X - Y is explained in the description of deletion.)

◦ Examples:

Review examples 5 - 9, 17, 18 in the preceding Chapter.

3. Retrieval

The value of $A + B$ is the first node on the list linked to A by B, if such a list exists. If the list does not exist, then $A + B$ evaluates to A. The success of retrieval is, as usual, open to testing as specified by the ti in the syntax + id ti (or bo id ti) of suff or dsuff. Retrieval of a Hollerith SEQ block has as its value the contents of the first block cell. The COMMON variable LVHOL is used to distinguish SEQ and non-SEQ storage. For SEQ Hollerith storage, LVHOL contains the SEQ index of the head block cell; for non-SEQ storage, LVHOL contains a zero.

◦ Examples:

Review examples 11 - 13 in the preceding Chapter.

4. Deletion

The effect of $A - B$ is to delete any list that might have been linked to A by B. The operation is said to have "failed" only if there was no list to have been deleted. Again, the success or failure may be detected by test. The value of $A - B$ is the value of $A + B$.

◦ Example:

See example 16 in the preceding Chapter.

5. Comparison

The effect of $A = B$ is to succeed if A and B have the same value. Similarly, the effect of $A \# B$ is to succeed if A and B do not have the same value. The value of either of these operations is A. Note that B need not be an identifier. For example, if A happens to have the value "3", then $X \ Y \ A = "3" //TAG$ will go to TAG. Actually, $X \ Y \ A = 3 //TAG$ is just as valid.

◦ Example:

Review examples 17 - 28 in the preceding Chapter.

◦ Examples:

Review examples 12-14, 16 in the preceding Chapter.

6. Inclusion

The effect of $A+B:C$ is to succeed if C occurs at least once on the list linked to A by B . The value of $A+B:C$ is the list index at which C first occurs. If it fails, the value is the value of $A+B$.

◦ Examples:

Review examples 28-30, 32 in the preceding Chapter.

7. Indication

Executing an indication (ix) after the link identifier of a retrieval or insertion, accesses the item of stipulated index from the linked list under question. Thus, $A+B+.3$ means: retrieve the third item on the list linked to A by B ; similarly, $A B+.3C$ means: do $A B+.3$ and place C on the list between the third item retrieved and its predecessor. The '+' signs surrounding the '.' may and usually are replaced by emp. If the '+' preceding the '.' is replaced by '-', the meaning is changed from retrieval to deletion. Thus, the destructive insertion $A B-.3C$ means: delete the third item on the A,B list ($A=B-.3$) and then make C the new third item ($A B.2C$). If the plus (+) following the period (.) is replaced by minus (-), the indicated item is found by counting up from the bottom of the list. Thus, for a list of length 5, $A+B.2$ and $A+B.-3$ retrieve the same item. Counting up or down the list can also be done by index type (ixtyp). Thus, $A+B=5$, $A+B./5$, $A+B.5$ means: retrieve the fifth identifier, Hollerith, numeric item, respectively from the A,B list.

When an index deletion is performed as in $X+Y-.I$, the value that remains is the deleted one. If there was no such value, then a failure is reported and X is left as the value. Similarly, in an index insertion such as $X Y .I Z$, the operation succeeds only if the list already has $I-1$ values or more. The index operation may be tested for this failure, and Z in any event is left as value.

◦ Examples:

Review examples 12, 13, 19-24, 28, 30, 31, 33 in the preceding Chapter.

PARENTHESES STATEMENTS

◦ Syntax:

Since the BNF syntax for parenthesized statements is complicated and illegible, we relegate it for completeness' sake to the Appendix. Instead, we introduce some new metasyntactic notation that properly fits our intuition about how parentheses are applied to a language that scans strictly from left to right: namely as nests for suffix sequences. Thus, the notation $X(Y$ will mean

$$\begin{array}{l} X(Z) \mid X Y \\ Z ::= Y,Z \mid Y \end{array}$$

That is, the parenthesis permits an optional, repeated sequence of that which follows the parenthesis. If X is empty in the case $W(X(Y$, then the second parenthesis option is void. The following syntax is a direct modification of unpar in the preceding section.

par ::= unpari(psuff | unpari(pdsuff)) (38)

unpari ::= unpar | id (39)

psuff ::= id ti(ti pm(pix ti(ti dseq id ti(ti psuff
| +(id ti(ti pm(pixn ti(ti psuff
| bo(id ti(ti psuff
| '(idf ti(ti psuff
| usemp (40)

usemp ::= usuff | emp (41)

pdsuff ::= id ti(ti pm(pix ti(ti dseq data ti weq
| +(id ti(ti :(obj ti weq
| psuff(pdsuff) (42)

pixn ::= pixo(ivc
| pixo:(obj) (43)

pix ::= pixn | emp (44)

pixo ::= .(pm(ixtyp
| .(ixtyp(pm) (45)

weq ::= (idf(weq
| eqn(id ti(ti weq
| emp) (46)

eqn ::= = | #

◦ Semantics:

The formidable syntax given in the Appendix has a very simple interpretation: namely, that any prefix fragment of a statement may be followed by a parenthesized sequence of suffixes, any one of which in turn may be interrupted to establish a prefix fragment to further parenthesized sequences. Some minor qualifications are necessary. First, parenthesization must not contain the empty string. Thus, (), ((A B C)), and A((B C)) are illegal. There would be no difficulty in handling these forms, but it seems to be a better debugging aid to spot useless parenthesis. Secondly, certain categories quite reasonably do

not take suffixes. For example, dseq can only prefix a single node, namely the first item of a SEQ block. Similarly, the symbols of a transfer operation occur only once for any given test. Finally, data suffixes (dsuff) may be prefix only to identification and or comparison sequences.

- Example:

The preceding Chapter is replete with examples of the parenthesized statement.

GIRL STATEMENT

- Syntax:

GIRL statement ::= unpar | par

IV. IMPLEMENTATION AND OPERATIONAL REQUIREMENTS

GIRL/FORTRAN TRANSLATION

A GIRL/FORTRAN program is translated to machine code in two steps:

- a preprocessing phase which translates the GIRL code to sequences of FORTRAN subroutine calls and stack manipulations.
- a FORTRAN compilation of the entire program.

A one-pass GIRL/FORTRAN compiler is under consideration.

MEMORY ALLOCATION

In the preprocessing stage, FORTRAN subroutine calls are assembled to interface an efficient associative memory simulation called GIRS⁷ (Graph Information Retrieval System). GIRS is currently implemented on the CDC 6700 as a hashed-address, direct-chained memory scheme. In the packed version of GIRS, insertion of a node-link-node triple requires one 60-bit word; insertion of a list of N triples requires N+1 such words for N>1. In the unpacked version of GIRS, each triple requires four 60-bit words; each list of N triples requires 4N+4 such words for N>1. The size of the graph—i.e., the number of available node addresses—may in no case exceed 2^{15} , and in general will be limited to a lower bound by the amount of core made available to the user. A graph paging facility is now under development.

⁷ Berkowitz, S., "Design Trade-Offs for a Software Associative Memory," Naval Ship Research and Development Center Report 3531 (May 1973).

In addition to the graph memory as such, one may also use a sequential memory for more rapid access to sink data lists. For this purpose, one must declare a vector SEQ in a DIMENSION or COMMON statement.

One may compress or expand the hashed-address memory of a previously constructed graph. The deck setups for accomplishing this task are given in a later section.

On the CDC 6700, the preprocessor binary program requires a total of 6736 words, including a 2416 word buffer. During FORTRAN compilation and execution, the preprocessor is overlaid and the core memory field length reduced or expanded to meet the program requirements. During execution, the GIRS binary program occupies 1607 words for the unpacked version, and 2592 words for the packed version.

GIRS FUNCTION EXECUTION TIMES

Because of the structure of the CDC 6700 operating system, it is difficult to give accurate estimates of average execution times. The following table, therefore, indicates very approximate times for executing INSERT(A-B-C), FIND(A+B), and DELETE(A-B) functions.

AVERAGE CDC-6600 EXECUTION TIMES FOR GIRS ROUTINES
(in microseconds)

	Unpacked		Packed	
	Minimum	MVL Increment	Minimum	MVL Increment
INSERT	60	0	185	0
FIND	50	15	76	28
DELETE	63	28	192	169

The tables lists minimum execution times and multivalue increment times for the packed and unpacked versions. The minimum times refer to operations on lists of a single node-link-node triple in the absence of hash conflicts. When the memory is ρ percent full, one may expect hash conflicts to increase the minimum times by an increment of 12.5ρ microseconds as a very rough rule of thumb. For lists of more than one triple, the multivalue (MVL) increment time for INSERT represents is zero since the bottom of the list is accessed uniformly by an up-pointer for lists of any length. For FIND or DELETE, the MVL increment indicates the additional time required to retrieve the n^{th} list item or to delete a list of n values, as opposed to retrieving the $n-1^{\text{th}}$ item or to deleting a list of $n-1$ values, respectively.

Two final comments on the table are in order. First, the average minimum time for FIND when the retrieval fails is 32 microseconds for either the packed or unpacked version. Secondly, the MVL increments for the FIND routine may be substantially reduced by use of a saved index option. Indeed, the retrieval times for any item on a list in an iterative search with saved index are a constant 103 and 141 microseconds for the unpacked and packed versions, respectively. Basically, the saved index mechanism retains the last accessed list address for each index variable in the GIRL code so that each new access search does not have to begin from the head of the list. Thus, in iterating down a list of k items, one can expect k increments rather than $k(k+1)/2$. If items are added to or deleted from the list body, there is the danger that subsequent indexes references to the list will be out of sequence if they have retained a previous saved index address. There is a mechanism imbedded in the preprocessor for circumventing this danger, but it is somewhat bulky. Moreover, operations

on short lists perform more efficiently without the overhead of a saved index option. Therefore, one may void the saved index option either for the entire program—by placing NOSAVE on the options control card, as described in the next section—or for part of the program—by setting the distinguished variable LVNOSV equal to unity at any point in the program. Setting LVNOSV=0 restores the saved index option.

GIRL DECK SETUPS

In the following, the CDC 6700 batch-entry deck setups for GIRL/FORTRAN runs and for memory compress/expand runs are presented. Lower case letters indicate card images that are either described after the setup or are user dependent (job card, charge card). Upper-case letters refer to character images. All system card images begin in Column 1. Those which are indented refer to FORTRAN card images whose code begins in Column 7. The card number is for reference only and need not be typed.

Batch-Entry Deck Setup for a GIRL/FORTRAN Program

	<u>Card No.</u>
job card	(1)
charge card	(2)
ATTACH, PREP, CAIZPREPBIN, MR=1.	(3)
ATTACH, GIRS, CAIZGIRSBIN, MR=1.	(4)
PREP.	(5)
FIN, I=TAPE8.	(6)
FTN. (used only if purely FORTRAN routines are to be run)	(6a)
LOAD, LGO.	(7)
GIRS.	(8)
end of record	(9)
memsize option1 option2 ...	(10)
PROGRAM name	
or	}
\$ SUBROUTINE name	(11)
non-DATA specification statements	:
G DEFINE string (optional)	(12)
DATA string (optional)	(13)
G EXECUTE	(14)
GIRL/FORTRAN executable code (no END statement)	:
G COMPLETE	(15)
other GIRL/FORTRAN routines	:
/ COMPLETE	(16)
end of record	(17)
purely FORTRAN routines	:
end of record	(18)
data	:
end of record	(19)
end of file	(20)

Notes:

1. End of record is accomplished by a simultaneous 7/8/9 punch in Column 1. End of file is accomplished by a simultaneous 6/7/8/9 punch in Column 1.
2. In the GIRL/FORTRAN program, GIRL statements are declared by punching a G in Column 1. Continuation cards are handled as in FORTRAN.
3. The options Card[†] (10) has the following entries:
 - memsize - The first six columns store an integer of at most six digits that stipulates the number of possible nodes that the graph may contain. There is no default; some integer must be entered.
 - *IXX - An integer of at most two digits preceded by an asterisk (*) declares the file number on which an old graph is stored. Default implies the setup of a new graph.
 - \$IIIIII - An integer of at most six digits preceded by a dollar sign (\$) declares the size of SEQ. Default size is one location.
 - PACK - Sets up code for packed version of GIRL. Default is unpacked version.
 - PRINT - Prints GIRL program on output file. Default is no-print.
 - COMMENTS - Places GIRL code with a C in Column 1 into preprocessed FORTRAN code. Default is no-comment
 - NOSAVE - Eliminates 'saved index' facility, and is therefore appropriate for short multivalued lists. See the discussion of 'saved index' in the previous section.

[†] Except for the first entry (first six columns), the other entries are optional and may appear in any order, separated by at least one space or comma.

Batch-Entry Deck Setup For Cataloguing a Graph Prior to Compression or Expansion

Compression or expansion of graph memory leads to a re-ordering of node addresses relative to the cell address at which node-link-node triples are stored. Therefore, one must save node addresses (which are of special interest to programs that massage the graph) so that the compress/expand program (described in the next subsection) can retrieve the node address mappings. In the following setup, these addresses are represented by `var1, ..., varn`. The metastymbol pfname refers to a permanent file name to be assigned by the user.

The deck setup is the same as for any GIRL/FORTRAN run with the following additions.

- Card (4) is followed by
REQUEST TAPE17, *PF.
- Card (8) is followed by
CATALOG, TAPE17, pfname.
- Card (10) should include the option PRINT.
- Card (11) should have (TAPE17,...) following the PROGRAM name, where the dots indicate other files used by the program.
- Card (15) should be preceded by
CALL GIRSDMP(0,0,17)
WRITE(17) n, var1, ..., varn

Batch-Entry Deck Setup for Graph Memory Compression or Expansion

The deck setup is the same as for any GIRL/FORTRAN run with the following additions:

- Card (4) is followed by
ATTACH, TAPE99, pfname.
REQUEST, TAPE27, *PF.

- Card (8) is followed by
CATALOG, TAPE27, pname.
- Card (10) should include /IIIIII—an integer of at most six digits preceded by a slash, declaring the size of the new graph memory. The card should also include *99.
- Card (11) should have (TAPE99, TAPE27) following the PROGRAM name. The deck is completed by

```

COMMON /LSAVE/ n, var1,..., varn
G EXECUTE
  READ(99) n, var1,..., varn
  CALL CONVERT
  CALL LVDUMP(0,0,27)
  WRITE(27) n, var1,..., varn
G COMPLETE
/ COMPLETE
end of record
end of file

```

NOTATION

The notation used in the preceding Chapters will be used for the CDC 6700 implementation language of GIRL. The publication language notation is proposed to be the same except for the changes indicated in the following table:

CDC 6700 Implementation Language	Publication Language
#	≠
i	≡
.=	.≡
/	?
//	!
:	∪
" "	' '

CHANGES IN THE MANUAL

It is expected that the manual format, programmed learning Chapter, and implementation will be modified and/or expanded from time to time. Those who wish to be placed on the mailing list to receive such changes should write the author. Suggestions and criticisms are welcome. Information related to actual use of the language would be appreciated.

ACKNOWLEDGMENTS

The language described in this report was implemented on the CDC 6700 by I. Zaritsky. His comments and those of G. Gluck and J. Garner are gratefully acknowledged.

APPENDIX A
EXPRESSIONS IN GIRL

EXPRESSIONS IN GIRL*

No.	Form	Value on Failure/Success	Expression Name	Meaning
1.0	X+Y	X/X+Y	Retrieval	Find the top of the sink node <u>List</u> connected to source node X by the link Y.
1.1	X+Y.INDEX	X+Y/ X+Y.INDEX (Note 1)**	Indication	Find the INDEX th item on the list connected to X by Y (cf. 4.0 FOR INDEX).
1.2	X+Y :DATA	X+Y/INDEX (Note 1)	Inclusion	Find the INDEX of the first occurrence of DATA on the list connected to X by Y (c.f. 4.0 for INDEX, 5.0 for DATA.)
2.0	X-Y	X+Y/X (Note 2)	Deletion	Delete list connected to X by Y.
3.0	X Y DATA	DATA/DATA (Note 2)	Insertion	Connect X by Y to DATA (cf. 5.0 for DATA).
3.1	X Y .INDEX DATA	DATA/DATA (Note 2)	Indexed Insertion	Place DATA in INDEX th location on list connected to X by Y (cf. 4.0 for INDEX, 5.0 for DATA).
3.2	X Y :DATA1 DATA2	X/DATA2 (Note 3)	Inclusion Insertion	X(+Y/FAIL :DATA1/FAIL,Y, DATA2) (cf. 5.0, 8.0, 1.2, 1.1, 3.0, 11.0').
3.3	X Y :DATA1 INDEX DATA2	X/DATA2 (Note 3)	Inclusion Indexed Insertion	X(+Y/FAIL :DATA1/FAIL,Y.INDEX DATA) (cf. 4.0, 5.0, 8.0, 1.2,1.1., 3.0, 11.0).

* In the table, X may represent a source node or a prefix string whose value is an internal node address.

** The notes are collected at the end of the table.

EXPRESSIONS IN GIRL—Continued

No.	Form	Value on Failure/Success	Expression Name	Meaning
3.4	X Y -DATA	DATA/DATA (Note 2)	Replacement	X(-Y/FAIL,Y DATA)(cf. 5.0, 2.0, 8.0, 3.0, 11.0).
3.5	X Y :DATA;-DATA2	X/DATA2 (Note 3)	Inclusion Replacement	X(+Y/FAIL :DATA1/FAIL,-Y,Y DATA2) (cf. 1.2, 8.0, 2.0, 3.0, 11.0).
3.6	X Y;I1:I2 DATA	DATA/DATA (Note 2)	Sequential Space Insertion	Connect X by Y to DATA placed in SS at SEQ(I1+1); SEQ(I1) = I2.
4.0	$\left\{ \begin{array}{l} \emptyset \\ - \\ + \\ \emptyset \end{array} \right\} \cdot \left\{ \begin{array}{l} \emptyset \\ - \\ + \\ \emptyset \end{array} \right\} / \text{integer}$ <p>where integer may be constant variable "expression" "statement" :DATA (cf. 1.1 and Note 4)</p>	(cf. 1.1)	Index	Find (\emptyset) or delete (-) the INDEXed any (\emptyset), identifier (\emptyset), Hollerith (\emptyset), or numerical (\emptyset) item counting from the top (\emptyset or +) or bottom (-) of the list. The INDEX itself is an integer constant, variable, "expression", "statement", or :DATA.
5.0	identifier '/integer/Hollerith' '//no-quote Hollerith' "arithmetic expression or statement"	-	Data	Sink node DATA. Only the identifier is non-terminal.

EXPRESSIONS IN GIRL—Continued

No.	Form	Value on Failure/Success	Expression Name	Meaning
6.0	X 'NAMEY	-	Identification	Give the name NAMEY to X.
7.0	X=Y	X/X	Comparison	Do X and Y refer to the same address?
8.0	X/F/S	-	Conditional transfer	Go to label F if preceding operation fails; otherwise to S. F or /S may be null in which case control passes to succeeding string in the null option.
9.0	DEFINE X,Y,Z,	-	Definition	\$'X; \$'Y; \$'Z, etc. where \$ denotes a random address.
10.0	*FUNC(ARG,...)	-	Function	GIRL/FORTRAN function call with single address value.
11.0	X(, ,)	-	Suffix Sequence	String suffixes are contained in parentheses and separated by commas. Each suffix complements the prefix of the left parenthesis.

Note 1. If X+Y does not exist, then X is the failure value and an error message is sent.

Note 2. Failure indicates X+Y did not exist.

Note 3. Failure indicates :DATA did not exist. (If X+Y fails, it holds an empty list which clearly contains no DATA.) In any event, DATA and INDEX may be tested separately for success or failure.

Note 4. Indication has the forms in the following table.

	Retrieve		Delete	
	Down	Up	Down	Up
Any	.I	.-I	-.I	-.I
Integer	..I	..I	..I	..I
Hollerith	./I	./I	./I	./I
Identifier	.+I	.-I	.-I	.-I

APPENDIX B

GIRL MNEMONICS CARDS

- A. Retrieval (0,1,4,6,7,8,9,10)
 $X + B_n$
- B. Deletion (0,2)
 $X - B$
- C. Insertion (0,3,4,5,6,7,8,9,10)
 $X B_n a_p$
- D. Identification (0,11)
 $X \text{ NAMEY}$

NOTES

- X is a prefix string.
- Retrieval A+B means: find top of sink node list linked to source node A by link B.
- Deletion A-B means: delete sink node list linked to A by B.
- Insertion A B C means: connect A by B to C.
- Indication/inclusion a is null or indicates list item index (n · ω c I) or requests (:p) index of p if p is on list.
- Indication/inclusion b is a or an index inquiry :p n · ω c I.
- Destruction n is null (or +) or - (to delete the succeeding indexed item).
- Index type e is null (to allow any type) or:
 (i) = (identifier)
 (ii) / (Hollerith)
 (iii) . (number)
- Direction e is null or + (either of which searches up) or - (searches down).

- A. Retrieval (0,1,4,6,7,8,9,10)
 $X + B_n$
- B. Deletion (0,2)
 $X - B$
- C. Insertion (0,3,4,5,6,7,8,9,10)
 $X B_n a_p$
- D. Identification (0,11)
 $X \text{ NAMEY}$

NOTES

- X is a prefix string.
- Retrieval A+B means: find top of sink node list linked to source node A by link B.
- Deletion A-B means: delete sink node list linked to A by B.
- Insertion A B C means: connect A by B to C.
- Indication/inclusion a is null or indicates list item index (n · ω c I) or requests (:p) index of p if p is on list.
- Indication/inclusion b is a or an index inquiry :p n · ω c I.
- Destruction n is null (or +) or - (to delete the succeeding indexed item).
- Index type e is null (to allow any type) or:
 (i) = (identifier)
 (ii) / (Hollerith)
 (iii) . (number)
- Direction e is null or + (either of which searches up) or - (searches down).

- A. Retrieval (0,1,4,6,7,8,9,10)
 $X + B_n$
- B. Deletion (0,2)
 $X - B$
- C. Insertion (0,3,4,5,6,7,8,9,10)
 $X B_n a_p$
- D. Identification (0,11)
 $X \text{ NAMEY}$

NOTES

- X is a prefix string.
- Retrieval A+B means: find top of sink node list linked to source node A by link B.
- Deletion A-b means: delete sink node list linked to A by B.
- Insertion A B C means: connect A by B to C.
- Indication/inclusion a is null or indicates list item index (n · ω c I) or requests (:p) index of p if p is on list.
- Indication/inclusion b is a or an index inquiry :p n · ω c I.
- Destruction n is null (or +) or - (to delete the succeeding indexed item).
- Index type e is null (to allow any type) or:
 (i) = (identifier)
 (ii) / (Hollerith)
 (iii) . (number)
- Direction e is null or + (either of which searches up) or - (searches down).

- E. Comparison (0,12)
 $X = \text{NAMEY}$
- F. Conditional Transfer (0,13)
 $X / F / S$
- G. Random Node/Link (14)
 $\$$
- H. Suffix Sequences (0,15)
 $X (+B, C D, E) F+G, H-I$

- Index I is a positive or negative integer-valued constant, variable, "expression", "statement", index request :p.
- Sink Node Content p is an identifier, '/integer/Hollerith', '/no-quote Hollerith', "arithmetic expression or statement"
- A NAMEY means: give name NAMEY to node A.
- A=NAMEY means: do A and NAMEY refer to same node?
- /F/S means: go to label F if preceding operation fails; otherwise to S. F or /S may be null in which case control passes to succeeding string in the null operation.
- \$ means: generate a random address.
- The string means: X+B; X C D; X E F+G; X E H-I.

- E. Comparison (0,12)
 $X = \text{NAMEY}$
- F. Conditional Transfer (0,13)
 $X / F / S$
- G. Random Node/Link (14)
 $\$$
- H. Suffix Sequences (0,15)
 $X (+B, C D, E) F+G, H-I$

- Index I is a positive or negative integer-valued constant, variable, "expression", "statement", index request :p.
- Sink Node Content p is an identifier, '/integer/Hollerith', '/no-quote Hollerith', "arithmetic expression or statement"
- A NAMEY means: give name NAMEY to node A.
- A=NAMEY means: do A and NAMEY refer to same node?
- /F/S means: go to label F if preceding operation fails; otherwise to S. F or /S may be null in which case control passes to succeeding string in the null operation.
- \$ means: generate a random address.
- The string means: X+B; X C D; X E F+G; X E H-I.

- E. Comparison (0,12)
 $X = \text{NAMEY}$
- F. Conditional Transfer (0,13)
 $X / F / S$
- G. Random Node/Link (14)
 $\$$
- H. Suffix Sequences (0,15)
 $X (+B, C D, E) F+G, H-I$

- Index I is a positive or negative integer-valued constant, variable, "expression", "statement", index request :p.
- Sink Node Content p is an identifier, '/integer/Hollerith', '/no-quote Hollerith', "arithmetic expression or statement"
- A NAMEY means: give name NAMEY to node A.
- A=NAMEY means: do A and NAMEY refer to same node?
- /F/S means: go to label F if preceding operation fails; otherwise to S. F or /S may be null in which case control passes to succeeding string in the null operation.
- \$ means: generate a random address.
- The string means: X+B; X C D; X E F+G; X E H-I.

GIRL
GRAPH INFORMATION RETRIEVAL LANGUAGE
FOR THE CDC 6400/6600/6700

Naval Ship Research and Development Center
Pattern Recognition Research Group
Code 1834
Bethesda, Maryland 20034

GIRL
GRAPH INFORMATION RETRIEVAL LANGUAGE
FOR THE CDC 6400/6600/6700

Naval Ship Research and Development Center
Pattern Recognition Research Group
Code 1834
Bethesda, Maryland 20034

GIRL
GRAPH INFORMATION RETRIEVAL LANGUAGE
FOR THE CDC 6400/6600/6700

Naval Ship Research and Development Center
Pattern Recognition Research Group
Code 1834
Bethesda, Maryland 20034

APPENDIX C
PARENTHESES STATEMENT BNF SYNTAX

```

par      ::= unpar i s
unpar i ::= unpar | id
usemp   ::= usuff | emp
s       ::= (sa) | w
sa      ::= sa, sa | ti w | ti usuff
w       ::= id ti wpa | + wpb | bo wpc | ' wpd

wpa     ::= (sb) | wa
sb      ::= sb, sb | ti wa | ti pm ix ti cu
cu      ::= dseq id ti usemp | dseq data ti
wa      ::= pmn wpe | . wpf | cp
cp      ::= dseq id ti s | dseq data ti weq

wpe     ::= (sc) | wb
sc      ::= sc, sc | wb | . c ca cu | cu
c       ::= pm ixtyp
ca      ::= ivc ti | :obj ti
wb      ::= . wpf | cp

wpf     ::= (sd) | wc
sd      ::= sd, sd | wc | c ca cu
wc      ::= pmn wpg | ixtypn wph | wd
wd      ::= ivc ti upi | : wpj

wpg     ::= (se) | we
se      ::= se, se | we | ixtyp ca cu
we      ::= ixtypn wpk | wd

wph     ::= (sf) | wf
sf      ::= sf, sf | wf | pm ca cu
wf      ::= pmn wpk | wd

wpi     ::= (sq) | cp
sq      ::= sq, sq | ti cp cu | cu

```

wpj ::= (sh) | wg
 sh ::= sh, sh | wg | obj ti cu
 wg ::= obj ti wpi

 wpk ::= (si) | wd
 si ::= si, si | wd | ca cu

 weq ::= (seq) | wea
 seq ::= seq, seq | ti wea
 weqe ::= weq | emp
 wea ::= eqn web | ' wec
 web ::= (seb) | divc ti weqe
 seb ::= seb, seb | divc ti weqe
 wec ::= (sec) | idf ti weqe
 sec ::= sec, sec | idf ti weqe

 wpb ::= (sj) | id ti wpl
 sj ::= sj, sj | id ti wpl | id ti cbu
 cbu ::= pm ixn ti usemp | : obj ti

 wpl ::= (sk) | wh
 sk ::= sk, sk | ti wh | ti cbu
 wh ::= pmn wpm | . wpm | : wpo

 wpm ::= (s1) | . wpm
 s1 ::= s1, s1 | . wpm | ixn ti usemp

 wpn ::= (sm) | wi
 sm ::= sm, sm | wi | c ca usemp
 wi ::= pmn wpp | ixtypn wpq | cbp
 cbp ::= ivc ti s | : wpr

 wpo ::= (sn) | obj ti weq
 sn ::= sn, sn | obj ti weq | obj ti

 wpp ::= (so) | wj
 so ::= so, so | wj | ixtyp ca usemp
 wj ::= ixtypn wpc | cbp

wpq ::= (sp) | wk
 sp ::= sp, sp | wk | pm ca usemp
 wk ::= pmn wps | cbp

 wpr ::= (sq) | obj ti s
 sq ::= sq, sq | obj ti s | obj ti usemp

 wps ::= (sr) | cbp
 sr ::= sr, sr | cbp | ivc ti usemp | : obj ti usemp

 wpc ::= (ss) | id ti s
 ss ::= ss, ss | id ti usemp

 wpd ::= (st) | idf ti s
 st ::= st, st | idf ti usemp

INITIAL DISTRIBUTION

Copies		Copies	
1	DODCI T. Braithwaite	5	NAVPGSCOL 1 M. Woods 1 D. Williams 1 G. Barksdale 1 C. Comstock
1	ARPA L. Roberts		
2	U.S. Army Picatinny Arsenal 1 R. Isakower	1	NAVWARCOL
1	U.S. Army Frankfort Arsenal D. Frederick	1	USNROTC & NAVADMINU, MIT
1	USAMERDC J. Marburger	1	NAVCOSSACT
4	CNO 1 OP 916 1 OP 916C1, LCDR Poteat 1 OP 916D 1 OP 098TD, L. Aarons	1	ADPESO
1	CMC	1	CGMCDEC
6	CHONR 1 400R, R. Ryan 1 430, R. Lundegard 1 432, L. Bram 1 437, M. Denicoff 1 437, G. Goldstein	1	ONR Boston
1	DNL	1	ONR Chicago R. Buchal
8	CHNAVMAT 1 MAT 0141E, R. Jeske 1 MAT 03 1 MAT 03A, CDR Booth 1 MAT 03L, J. Lawson 1 MAT 03L4, J. Huth 1 MAT 03P2, P. Newton 1 MAT 03P21, S. Atchison	1	ONR Pasadena R. Lau
4	USNA 1 D. Rogers 1 A. Adams 1 Dept of Math	5	NRL 1 5030, S. Wilson 1 5400, B. Wald 1 7810, A. Bligh 1 8050, CDR Tatro
		1	COMNAVINT
		1	NAVELECSYSCOM
		7	NAVSHIPSYSCOM 1 SHIPS 03, RADM Andrews 1 SHIPS 0311, B. Orleans 1 SHIPS 03414, A. Chaikin 1 SHIPS 03423, C. Pohler 1 SHIPS 0719, L. Rosenthal 1 SHIPS 08, Nuclear Power Directorate
		3	NAVAIRSYSCOM 1 NAVAIR 5033, R. Saenger 1 NAVAIR 5333F4, R. Entner 1 NAVAIR 5375A, J. Polgren
		1	NAVFACENCOM

Copies

1 NAVORDSYSCOM
 1 NAVAIRDEVCEEN
 1 CIVENGLAB
 10 NELC
 3 5000, A. Beutel
 3 5200, M. Lamendola
 3 5300, J. Dodds
 1 NAVUSEACEN
 1 NAVWPNSCEN
 L. Diesen
 1 NAVCOASTSYSLAB
 2 NOL
 1 H. Stevens
 6 NWL
 1 Code K
 1 Code K-1
 1 Code KO
 1 Code KP
 1 Code KPS
 8 NAVSEC
 3 SEC 6102C, P. Bono
 1 SEC 6114, R. Johnson
 1 SEC 6114E, A. Fuller
 1 SEC 6178D03, L. Biscomb
 1 AFOSR
 Code 423
 1 Rome Air Development Center
 1 WPAFB AFFDL
 12 DDC
 1 NASA Langley Research Center
 R. Fulton
 1 Carnegie-Mellon University
 Professor A. Newell
 1 College of William & Mary
 Professor N. Gibbs

Copies

6 MIT
 1 Professor M. Minsky
 1 Professor T. Winograd
 1 Professor P. Winston
 1 Dr. A. Nevins
 1 D. McDermott
 1 G. Sussman
 1 MIT Lincoln Laboratory
 R. Rovner
 1 Ohio State University
 Professor L. White
 1 Southern Methodist University
 Professor R. Korfhage
 2 Stanford University
 1 Professor J. McCarthy
 1 Professor J. Feldman
 1 UCLA
 Professor M. Melkanoff
 1 University of Florida
 Professor J. Tou
 1 Hughes Research Laboratory
 B. Bullock
 1 IBM Federal Systems Division
 J. Sammet
 2 IBM Watson Research Laboratory
 Yorktown Heights, New York
 1 G. F. Codd
 1 C. H. Thompson
 2 Stanford Research Institute
 1 Dr. C. Rosen
 1 Dr. B. Raphael
 1 Systems Development Corporation
 Santa Monica, California
 E. Book
 1 Xerox Research Laboratories
 Palo Alto, California
 Dr. D. Bobrow

CENTER DISTRIBUTION

Copies	Code
1	18/1809
1	1802.1
1	1802.3
1	1802.4
2	1805
2	183
1	1832
1	1833
30	1834
1	1835
2	184
2	185
1	1858
2	186
1	1863
1	1867
2	188
2	189
2	1891, Central Depository

Security Classification

DOCUMENT CONTROL DATA - R & D

Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified

1. ORIGINATING ACTIVITY (Corporate author) Naval Ship Research and Development Center Bethesda, Maryland 20034		2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	
		2b. GROUP	
3. REPORT TITLE GRAPH INFORMATION RETRIEVAL LANGUAGE; PROGRAMMING MANUAL FOR FORTRAN COMPLEMENT			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates)			
5. AUTHOR(S) (First name, middle initial, last name) Sidney Berkowitz, Ph.D.			
6. REPORT DATE June 1973	7a. TOTAL NO. OF PAGES 57	7b. NO. OF REFS 0	
8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S) 4137		
b. PROJECT NO. SR0140301			
c. 1-1834-001	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)		
d.			
10. DISTRIBUTION STATEMENT			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY SHIPS 00311	
13. ABSTRACT GIRL (Graph Information Retrieval Language) is a programming language designed to conveniently manipulate information in graph structures. As such, the language will plan a key role in the construction of the organizational schemes found, for example, in information retrieval, pattern recognition problems, linguistic analysis, and process scheduling systems. The language is written to <u>complement</u> an algebraic language, in the sense that GIRL statements are distinguished from the statements of the algebraic language and the statements may be interleaved. The primary advantage of separating symbolic and numeric statements is that the programmer is afforded a linear, one-one trace of graph operations in the code description.			

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Programming languages Associative memory Graphs Lists						