1064

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication. (See GEI 13.9-3)

Title- **Agen - An Associative Memory Generator**

Date- **September 2, 1975**

TM- **75-1274-18**

Other Keywords- **Computer Languages**
Data Management

Author **M. E. Lesk** Location **MH 2C-569** Extension **6377**

Charging Case- **39199**
Filing Case- **39199-11**

ABSTRACT

Agen writes programs to store and retrieve information. Each stored item contains data and a key; to retrieve or allocate the item, just mention the key. Possible applications include symbol tables, sparse arrays, or table lookups of words. Two possible algorithms are built in: the user can request a binary tree search or a hash search form of storage organization.

Agen is a program generator, which translates specifications into a high-level programming language which can be mixed with the user's general purpose code. Each user request for a lookup procedure is translated into a subroutine to perform the search and allocation if needed. This subroutine may be viewed as if it addressed an array of pointers to the user's data items. Agen is available on GCOS and UNIX. It writes lookup subroutines in the C language.

Pages Text 9	Other 0	Total 9
No. Figures 1	No. Tables 1	No. Refs. 1

BELL TELEPHONE LABORATORIES, INC.

TM-75-1274-18

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

<AHO,A V
<BECKER,R A
<BOYCE,W M
<BROWN,W STANLEY
<CHAMBERS,J M
<COOPER,DENNIS W
<FLEISCHER,HERBERT I
<FRASER,A G
<GOLDSTEIN,A JAY
<GUTHERY,SCOTT B
<HAMILTON,PATRICIA
HAMMING,R W
+HANNAY,N B
<JOHNSON,STEPHEN C
<KEESE,W M
<KERNIGHAN,BRIAN W
<LUDERER,GOTTFRIED W R
<MARANZANO,JOSEPH F
<MC GILL,ROBERT
MC ILROY,M DOUGLAS
+MCDONALD,H S
MORGAN,S P
<OSSANNA,J F JR
PINSON,ELLIOT N
+PRIM,ROBERT C
<RALEIGH,THOMAS M
<ROBERTS,CHARLES S
TERRY,M E
<THOMSON,M-L
29 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

ABRAHAM,STUART A
ACKERMAN,A F
AHRENS,RAINER B
ALBERTS,BARBARA A
ALCALAY,DAVID
ALLISON,CHARLES E
ALT,MISS DOROTHY L
AMOSS,JOHN J
AMRON,I

COVER SHEET ONLY TO

ANDERSON,MS K J
ARMSTRONG,DOUGLAS B
ARNOLD,GEORGE W
ARNOLD,S L
ATAL,B S
BACCASH,MISS J M
BADURA,DENNIS C
BAKER,BRENDA S
BASEIL,RICHARD J
BAUER,MS H A
BAUGH,C R
BAYER,DOUGLAS L
BERNSTEIN,LAWRENCE
BERTH,R P
BEYER,JEAN-DAVID
BICKFORD,N B
BILLINGTON,MISS M J
BILOWOS,RICHARD M
BIRCHALL,R H
BIREN,MRS IRMA B
BISHOP,MISS V L
BLINN,JAMES C
BLUE,J L
BLUM,MRS MARION
BLY,JOSEPH A
BOCKUS,R J
BODEN,F J
BONANNI,LORENZO E
BORKIN,S A
BOURNE,STEPHEN R
BOWERS,J L
BOWYER,L RAY
BRADLEY,R H
BRANDT,RICHARD B
BROWN,COLIN W
BROWN,WILLIAM R
BUCHSBAUM,S J
BULLEY,RAYMOND M
BURG,F M
BURNETTE,W A
BURROWS,T A
BYRNE,EDWARD R
BZOWY,D E
CAMPBELL,J H
CAMPBELL,STEPHEN T
CANADAY,RUDD H
CARAWAY,R E
CARDOZA,WAYNE M
CARRAN,J H
CARR,DAVID C
CASPERS,MRS BARBARA E
CAVINESS,JOHN D
CHAFFEE,N F
CHAMBERS,MRS B C
CHEN,EDWARD
CHEN,STEPHEN
CHERRY,MS L L
CHIANG,T C
CHODROW,MARK M
CHRIST,C W JR
CLAYTON,D P
CLIFFORD,ROBERT M

COVER SHEET ONLY TO

COBEN,ROBERT M
COHEN,HARVEY
COKE,MISS E U
<COLE,LOUIS M
COLE,M O
COOK,THOMAS J
COOPER,A E
COPP,DAVID H
COREY,D A
COSTELLO,PETER E
COSTON,WALTER P
<COULTER,J REGINALD
CRAGUN,D W
CRANE,RODERICK P
CRUME,LARRY L
CUNNINGHAM,STEPHEN J
CUTLER,C CHAPIN
D ANDREA,MRS LOUISE A
D STEFAN,D J
DAVIS,D R
DAVIS,R D
DE JAGER,D S
DESENDORF,JUDITH
DEUTSCH,DAVID N
DICKMAN,B N
DIMMICK,JAMES O
DOLAN,MRS MARIE T
DOLOTTA,T A
DONOFRIO,L J
DOWD,PATRICK G
DRAKE,MRS L
DRUMMOND,R E
DUDLEY,MRS E H
DUFFY,FRANCIS P
EDELSON,D
EDMUNDS,T W
EIGEN,D
EITELBACH,DAVID L
ELLIOTT,R J
ELY,T C
ERDLE,K W
ERRICHIELLO,PHILIP M
ESSERMAN,ALAN R
ESTOCK,R G
FABISCH,MICHAEL P
FEDER,J
FELDMAN,STUART I
FELS,ALLEN M
FERIDUN,K K
FINUCANE,JOHN J
FISCHER,H B
FLYNN,MS M L
FONG,KENNETH T
FORTNEY,V J
FOUNTOUKIDIS,A
FOWLKES,E B
FOX,PHYLLIS
FOY,J C
FRANKLIN,DANIEL L
FRANK,H G
FRANK,MISS A J
FRANK,RUDOLPH J

COVER SHEET ONLY TO

FREEDMAN,M I
FREEMAN,K GLENN
FREEMAN,R DON
FREIDENREICH,MRS B
FROST,H BONNELL
FULTON,ALAN W
GAJEWSKA,MS HANNA O
GANNON,T F
GARCIA,R F
<GATES,G W
GAWRON,L J
GAY,FRANCIS A
GEARY,M J
GEPNER,JAMES R
GERGOWITZ,E B
GEYLING,F T
GIBB,KENNETH R
GIBSON,H T JR
<GILLETTE,DEAN
GIMPEL,JAMES F
GITHENS,JOHN A
GLASER,W A
GLASSER,ALAN L
GLUCK,F
<GNANADESIKAN,R
GOGUEN,MS NANCY
GOLABEK,MISS R
GOLDSMITH,L D
GORMAN,JAMES E
<GRAHAM,R L
GRAVEMAN,R F
GREENBAUM,H J
GREENSPAN,S J
GROSS,ARTHUR G
GRUCHOWSKI,M P
GUERRIERO,JOSEPH R
HAFER,E H
HAIGHT,R C
HALE,A L
HALL,ANDREW D JR
HALL,MILTON S JR
HALL,W G
HALPIN,THOMAS
HAMILTON,MRS L
HARKNESS,C J
HARRISON,NEAL T
HARRIS,CHARLES S
HARUTA,K
HAUSE,A D
<HAWKINS,DONALD T
HEATH,SIDNEY F III
HELD,RICHARD W
HERGENHAN,C B
HEROLD,JOHN W
HESS,MILTON S
HOEHN,MISS MARIE J
HOLTMAN,JAMES P
HONIG,W L
HOYT,WILLIAM F
HUDSON,E T
HUNNICUTT,CHARLES F
HUPKA,MRS FLORENCE

COVER SHEET ONLY TO

HYMAN,B
IFFLAND,FREDERICK C
IMAGNA,CLYDE P
IPPOLITI,O D
IRVINE,M M
<IVIE,EVAN L
JACKOWSKI,D J
JACOBS,H S
JAKUBEK,RAYMOND J
JAMES,DENNIS B
JAMES,J W III
<JENSEN,PAUL D
JESSOP,WARREN H
JESSUP,RICHARD F
JOHNSTON,WALTER E JR
JONES,B R
<JUDICE,CHARLES N
KACHURAK,JOSEPH J
<KAISER,J F
KAMINSKY,MICHAEL L
KAPLAN,A E
KAPLAN,M M
KAUFMAN,LARRY S
KAYEL,R G
KEANE,E R
KELLY,L J
KENNEDY,ROBERT A
KERTZ,DENIS R
KEVORKIAN,DOUGLAS E
KILLMER,JOHN C JR
KLEIN,MISS R L
KNOWLTON,KENNETH
KNUDSEN,DONALD B
KNUDSON,DEAN O
KORNEGAY,R L
KREIDER,DANIEL M
LA CAVA,JOSEPH L
LAMBERT,MRS C A
LAYTON,RICHARD L
LEE,DENNIS F
LEGENHAUSEN,S
LESK,MICHAEL E
LESSEK,PETER V
LEVINE,P R
LICWINKO,J S
LIEBERT,THOMAS A
LINDERMAN,J
LIST,WILLIAM H S
LIU,T K
<LOGAN,MRS V G
LOMUTO,N
LONG,DAVID W
LORENC,ANTHONY
LUTZ,KENNETH J
<LYCKLAMA,HEINZ
LYONS,T G
MACHOL,R E JR
MACRI,PHILIP P
MADDEN,MRS D M
MAHLER,G R
MALCOLM,J A
MALLOWS,COLIN L
...
454 TOTAL

+ NAMED BY AUTHOR   > CITED AS REFERENCE   < REQUESTED BY READER   (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION...

COMPLETE MEMO TO:
127-SUP

COVER SHEET TO:
12-DIR   13-DIR   127

COPLGP = COMPUTING/PROGRAMMING LANGUAGES/GENERAL PURPOSE
UNPL# = UNIX/PROGRAMMING LANGUAGES

LESK,M E
MH 2C569

TM-75-1274-18
TOTAL PAGES 9

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.

PLEASE SEND A COMPLETE COPY TO THE ADDRESS SHOWN ON THE OTHER SIDE
NO ENVELOPE WILL BE NEEDED IF YOU SIMPLY STAPLE THIS COVER SHEET TO THE COMPLETE COPY.
IF COPIES ARE NO LONGER AVAILABLE PLEASE FORWARD THIS REQUEST TO THE CORRESPONDENCE FILES.

Bell Laboratories

Subject: Agen - An Associative Memory Generator
Case- 39199 -- File- 39199-11

date: September 2, 1975

from: M. E. Lesk

TM: 75-1274-18

MEMORANDUM FOR FILE

Agen is a program generator for table searches. It accepts descriptions of the table keys or indices and writes a subroutine to perform the appropriate lookup. Thus the programmer wishing to store information relating to words in a table can use a character string as a subscript, as if it were an integer.

Agen translates a file of descriptions of lookup routines, called the source file henceforth, into a file of programs implementing the lookups. The generated routines are written on a file named a.tab.c which can be compiled directly or included in a user's C program. The operation of Agen is shown in Figure 1.

Figure 1

Operation of Agen

```
 ________        __________________        _________
|        |      |                  |      |         |
| source | -->  | Agen translator  | -->  | a.tab.c |
|________|      |__________________|      |_________|


 _________       ____________       _________________
|         |     |            |     |                 |
| a.tab.c | --> | C compiler | --> | lookup routines |
|_________|     |____________|     |_________________|
```

Each routine is described by its name, the arguments it expects, and the kind of item stored. The lookup routines return pointers to the stored items. Whenever a new key is used, the storage for the new element appears automatically. Thus a typical source line for Agen might appear:

```
        word (string) st1;
```

indicating that the programmer wishes a table lookup function named "word", which has a string as its argument, and returns a pointer to items like "st1". The item stored is used to declare the size of the table entries, and may be replaced by the number of bytes to be stored. In the user's program, if "st1" is a structure declared like

```
        struct {int first, last, count;} st1;
```

then the program may contain operations like

```
        word("when")->first = 3;
        a = word("in")->last;
        if (word("the")->count > 0)
```

just as if there had been a declaration

```
        struct {int first, last, count;} word [ ];
```

and "when" or "in" could be magically converted to an appropriate integer subscript.

The kinds of subscripts accepted by Agen programs are integers, characters, and strings. Thus a function can be declared

```
        matrix (int, int, int) item;
```

to indicate three integer subscripts. Why is this better than

```
        matrix[a][b][c]
```

or some such normal multidimensional array? The difference is that the user need not know the values of a,b, and c; and the amount of storage used is proportional to the amount actually stored into, not the amount declared. In exchange, the fetch and store operations are much slower with Agen than with the normal hardware subscripting operations.

The operation of the Agen lookup routines is easy to describe. First, the arguments are converted to a single key. The key is the linear concatenation of all the _int_ or _char_ subscripts (all are received by the called program as _int_ because of C conventions) followed by the concatenation of all _string_ arguments (separated by '|' characters). This single key, which uniquely identifies the subscripts given, is now used in either a binary tree search or a hash search to find the appropriate element. If no such element exists, it is created, the storage assigned to it cleared, and a pointer to the new element returned. In the normal case, the lookup routine always returns a pointer to a valid item, whether previously defined or not. There can never be more than one item for a given key, since an element is created only if it can not be found in the table.

Sometimes this is not adequate. Thus, in addition to subscript type arguments, an inquiry type argument may also be used. This is specified as exist in an Agen source line. Such an argument is zero for normal operation; if it is supplied as 1 in a call, it indicates that the caller is merely inquiring as to the existence of an item under the requested subscripts. If the item exists, the usual pointer is returned. If it does not, zero is returned. Thus a definition

```
        word (string, exist) item;
```

if used with a call of the form

```
        word ("course", 0)
```

creates a new item is nothing is stored under the key "course" but the call

```
        word ("of", 1)
```

returns zero if "of" is not defined. This sort of argument has two other uses. If it is -1, it indicates deletion; that is,

```
        word("human", -1);
```

would delete an item stored as "human", if any existed. If the exist argument is 2, it indicates "find the next element". In this case, an element different from the one specified is returned, in such a way that continued requests for the "next" element will give the contents of the table. The pointer returned in this case is the pointer to the key, not the pointer to the value. An ordinary search will retrieve the value if desired. If the argument is a simple string or integer, and if the search is a binary tree search, note that the definition of the key is such as to return the table in lexicographic order. When an exist argument of 2 is given and the end of the table is reached, zero is returned.

Table 1 summarizes the uses of the "exist" argument.

There is one other kind of argument that can be supplied to an Agen lookup routine, called "size". Normally, all items stored by a lookup routine are the same size, and the size is given in the declaration. Often, however, it is desirable to store different sized items and specify every time a new item is created how big it is. In this case, a size argument must be declared to the lookup routine and given at the first use of each subscript. Changing the declaration of the "word" routine to

Table 1

The EXIST Argument

Value	Meaning
-1	Delete this entry if it exists. Return 0.
0	Normal operation; create if non-existent. Return a pointer to the data item.
1	Test existence. Do not create. Return a pointer to the data item if it exists, otherwise return 0.
2	Find another item. Return a pointer to the next data item (never the one specified by this key). If "no more" items, return 0.

```
        word (string, exist, size);
```

permits calls of the form

```
        word ("events", 0, 20);
```

indicating that if an item must be created for "events", it is to be 20 bytes long. Any existing item will be returned, regardless of length. This requires the maximum data size needed for a given key to be specified the first time the key is used, unless the item is deleted and recreated.

The _exist_ argument may be set to 2 to find the contents of a table. Successive calls will return different keys, which may then be used to obtain the data. With a binary tree search, "next" really means "next greater than ..." and thus the table can be obtaned in order of the keys. Note, however, the description above (page 2) of the way in which the keys are constructed from the arguments. A single integral or string key works as expected, but more complicated keys have the order of their constituents permuted to group integral and string subscripts. With hash searching, "next" means "next" in hash table order. Anything missing from the hash table returns the first item in the table as "next". Thus, no matter what the form of a table, the entries can be enumerated by successive probes with an _exist_ argument of 2 as follows:

1) use the null string or zero as a key;
2) use the returned key as the next key;
3) stop when zero is returned.

There are some interactions between the _exist_ and _size_ arguments if both are specified. If the _exist_ argument is 1 the _size_ argument is never relevant, since nothing can be created. If the _exist_ argument is -1, the _size_ argument is essential and _must_ give the size of the element to be deleted.

The source lines defining lookup routines also select the choice of a binary or a hash search. This is specified between the function name and its arguments. The user may give the name of the search method, and/or an estimated numerical length of the table. If the method specified is "hash", the length must be given. The default methods are binary tree search if the length is unknown and hash search if an estimated length was given. In general, hash searches are faster and use less storage. To define "word" as a hash-searched function, write

```
word 1500 (string, exist, size);
```

or

```
word hash 1500 (string, exist , size);
```

The default is the same as

```
word binary (string, exist, size);
```

The general syntax of Agen input is:

```
{source lines}
%%
{extra material}
```

Anything after a line containing just %% is copied to the output unchanged. So is anything contained between lines of %{ and %} as in Yacc and Lex. The format of a source line is

```
{routine name} {lookup type} {arguments} {item}
```

where _lookup type_ may be missing, or may be the word "binary" or the word "hash", and may contain a length as an integer. If the word "hash" is included, the length must be given. The _arguments_ are enclosed in parentheses or brackets and separated by commas. Each argument is one of the words _int_, _char_, _string_, _exist_, or _size_. The first three indicate subscripts to be used in specifying keys. The _size_ and _exist_ arguments control table handling and allocation.

Arguments may be given in any order. Only one instance of exist or size may appear; any number of subscripts can be used, in any order. The item indicates the size of the items (it can be omitted if there is a size argument) and should either be a number (taken to be in bytes) or the name of a variable to be inserted as an argument to the sizeof() function. Note that the variable has to be declared somewhere; this can either be done directly in the Agen source by use of %{ and %} to bracket the declaration, or in the file which contains a

```
# include "a.tab.c"
```

statement.

As an example, consider the problem of writing a program which reads a text and prints out a table showing the words in the text, the number of occurrences of each word, and the sequence number of the first occurrence. This is very similar to what would be needed to create a symbol table for a computer program, except that the data stored for each symbol would be different. Two source files will be required, one for Agen and one for Lex. Lex [1] is used to generate the program to isolate the words in the input stream. Also, the Lex routine name "yywrap" is used to print the table at the end of the input.

Agen source:

```
%{
     struct {int first, count;} datac;
%}
word (string, exist) datac;
%%
```

Lex source:

```
                int kw 0;
%%
[a-zA-Z]+       {
                ++kw;
                word(yytext, 0)->count++;
                if  (word(yytext, 0)->count == 1)
                     word(yytext, 0)->first = kw;
                }
.               ;
\n              ;
%%
yywrap()
{
char *s;
printf("Total First Word\n");
for (s = word("", 2); s != 0; s= word(s,2))
     printf("%5d %5d %s\n",word(s,0)->count,
          word(s,0)->first, s);
printf("total number of words was %d\n",kw);
return(1);
}
# include "a.tab.c"
```

The Agen source defines a table subscripted by character strings (words in this case) and noting for each word the point of first occurrence and the total number of occurrences. The Lex source uses a rule [a-zA-Z]+ to isolate strings of letters; each such word is looked up and its count incremented. If the count is one, the current word number _kw_ (incremented at each word) is stored in the _first_ field. When the end of input is reached, Lex calls _yywrap_; this routine runs through the table and prints the statistics for each word. Note the procedure for scanning the table: the string _s_ is initialized to the element found by taking the "next" item after the null string. Then successive "next" items are found until the end of the table is reached. Since the search method is a binary tree search, and the keys are simply single character strings, the "next" items are in alphabetical order. If a hash search had been used, the items would be in random order and would have to be sorted on output.

The appropriate commands to compile and run this are

```
agen agensource
lex lexsource
cc lex.yy.c -ll -lp
a.out <input
```

and if the input file is

```
when in the course of human events, it becomes
necessary for one people to dissolve the
political bands which have connected
them with another, and to assume among the
powers of the earth the separate and equal
station to which
```

The output is:

```
Total First Word
    1    28 among
    2    25 and
    1    24 another
    1    27 assume
    1    18 bands
    1     9 becomes
    1    21 connected
    1     4 course
    1    15 dissolve
    1    33 earth
    1    37 equal
    1     7 events
    1    11 for
    1    20 have
    1     6 human
    1     2 in
    1     8 it
    1    10 necessary
    2     5 of
    1    12 one
    1    13 people
    1    17 political
    1    30 powers
    1    35 separate
    1    38 station
    5     3 the
    1    22 them
    3    14 to
    1     1 when
    2    19 which
    1    23 with
total number of words was 40
```

As an example of timings, the above program was run under several options. As above, with a binary tree search and the portable library used for input and output, it processes the Declaration of Independence at 116 words per processor second. With the redundant searches eliminated by remembering a pointer to the just-made search, the speed is 160 words per processor second. By using a hash search and UNIX-tailored input and output routines, the speed can be raised to about 350 words per processor second.

In general, the hash searches are faster and require less memory than the binary tree searches. The main reason for using a binary tree search is that the user may be unwilling or unable to estimate the table size in advance. Hash tables are not grown; the size must not exceed 50% more than the original estimate. A single hash search takes somewhat over half a millisecond; a single binary tree search perhaps 30-50% more time. Even more unfortunate is that the storage for each item includes: a) the memory consumed by the item itself; b) the memory consumed by the key for the item; c) four extra words of memory for the binary tree search or two extra words for the hash search; and d) breakage in the allocator of two words in the case of hash searches and three words in the case of binary searches. When the items and keys are short, there is a lot of wasted storage.

Agen is available on GCOS as well as on UNIX. The speeds of GCOS programs are comparable to those of UNIX programs. To access Agen on GCOS, say

```
./agen sourcefile
```

and the _a_._tab_._c_ file is written as on UNIX.

There exist plans to improve Agen, by producing lookup routines more carefully tailored to the user's applications. This should improve the speed of search routines produced by Agen.

M E Lesk
M. E. Lesk

MH-1274-MEL-unix

References

[1] _Lex - A Lexical Analyzer Generator_, M. E. Lesk, 1975 TM 75-1274-15.