e Copy

ESD-TR-69-364

ESD ACCESSION LIST
ESTI Call No. 67503
Copy No. 1 of 2 cys.

MTR-927

HASP: A PL/1 HASH STORAGE PACKAGE

ESD RECORD COPY
RETURN TO
SCIENTIFIC & TECHNICAL INFORMATION DIVISION
(ESTI), BUILDING 1211

Joseph E. Sullivan

NOVEMBER 1969

Prepared for

DIRECTORATE OF PLANNING AND TECHNOLOGY
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

This document has been approved for public release and sole; its distribution is unlimited.

Project 512C
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract AF19(628)-68-C-0365

AD0697760

When U.S. Government drawings, specifications, or other doto ore used for any purpose other thon o definitely reloted government procurement operotion, the government thereby incurs no responsibility nor any obligotian whatsoever; and the foct thot the government moy have formulated, furnished, ar in any woy supplied the said drawings, specifications, or other data is not to be regorded by implication or otherwise, as in ony manner licensing the holder or any other person or corporotion, or conveying any rights or permissian to monufocture, use, or sell any patented invention that moy in any way be related thereto.

Do not return this copy. Retoin or destroy.

ESD-TR-69-364

MTR-927

HASP: A PL/1 HASH STORAGE PACKAGE

Joseph E. Sullivan

NOVEMBER 1969

Prepared for

DIRECTORATE OF PLANNING AND TECHNOLOGY
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

This document hos been approved for public release and sale; its distribution is unlimited.

Project 512C
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract AF19(628)-68-C-0365

FOREWORD

This report is the result of a need for hash coding schemes written in PL/1. Although the techniques have been available for some time, we believe this to be the first implementation of the technique in PL/1.

The work was performed under contract F19628-68-C-0365 for Electronics Systems Division, U.S. Air Force Systems Command.

REVIEW AND APPROVAL

This report has been reviewed and is approved.

WILLIAM F. HEISLER, Colonel, USAF
Chief, Command Systems Division
Directorate of Planning & Technology

ABSTRACT

A "generalized" PL/1 program is described which permits the assignment of "values" to a set of symbols, and the retrieval of the value assigned to a given symbol. That is, functions of character strings may be defined and evaluated. The method used is hash entry with chaining.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION I

INTRODUCTION

HASP is a PL/1 program package for general-purpose hash-coded storage and retrieval of symbolically indexed values. Ref. 1 provides general background on this subject; in its terms, the method used is a "scatter index table" with collisions resolved by direct chaining. This document is intended as a user's guide and reference.

SECTION II

PROGRAM USAGE

INITIALIZING (HASP)

Programming

For each "universe" (i.e., separate symbol table) that the user wishes to maintain, a structure of the general form

```
DCL  1  U,

         2  NE   BIN FIXED(31),
         2  HA   BIN FIXED(31),
         2  HN   BIN FIXED,
         2  HP   POINTER,
         2  SN   BIN FIXED,
         2  SI   BIN FIXED(31),
         2  SSP  POINTER,
         2  SP   POINTER,
         2  DV   BIN FIXED(31);
```

must be reserved (in any storage class) and retained at least while the table is in use. (Note that BASED storage class for U may be unfortunate for some purposes - see below). U is initialized by the statement

```
CALL HASP(U);
```

after the items HN, SN and probably HA, SI and DV have been set by the user. These and the other items are discussed individually below:

NE is the current number of items in the table. HASP sets NE to 0; NE is maintained by the HASP package thereafter.

HA is the hash entry algorithm number (see below). If not set to a legal number by the user prior to calling HASP, HASP sets it to 1. (Because 1 is the only legal choice at present, HASP sets HA to 1 unconditionally.) HA may not be modified while there are entries in the table.

HN is the size of the hash entry table (see below). With algorithm 1, HN must be an odd number - if it is not, HASP increments it by 1 with the comment

"HASP: HN SET TO ____".

HN must remain fixed during the life of the table.

HP is a pointer set by HASP to point to the hash entry (header) table. For many applications, the user need not be concerned with this item (other than not to modify it) nor the table it indicates. Instructions for gaining access to the header table in other cases are given below. HP is maintained by the HASP package.

SN is the current size, in bytes, of the symbol storage area (see below). It should be set by the user prior to calling HASP; this setting determines the initial allocation. Thereafter, SN is maintained by the HASP package.

SI is the amount, in bytes, by which the symbol storage area is to be expanded when the current allocation is exhausted. It can be set to any nonnegative value at any time by the user. However, if SI is found to be 0 by the HASP package at a time when more storage is needed, a condition called HSFULL is raised. If a normal recovery is programmed from this condition, it must include setting of SI to a positive value, e.g.,

```
ON CONDITION (HSFULL) SNAP SI=1000;
```

SSP and SP are pointers to a structure containing the symbol storage area and the area itself. As with HP, this would rarely by the user's worry - but see below if it is. SSP and SP are maintained by the HASP package.

DV is the "default value" for the space - the value returned when an undefined symbol is referenced. DV can be set at will by the user.

After initialization, a table is ready for addition, retrieval and deletion of entries. These functions are accomplished by other calls to the HASP package (actually additional entry points to the same program). The initialization entry is the only one which does not presume a prior initialized state for the table U; in all other cases, if an uninitialized table is referenced, an undefined result (IBMese for "a very messy error") will ensue.

The relationship among U, the header table, and the symbol storage area are illustrated in Fig. 1.

Hash Entry Algorithm

The efficiency of hash coding depends critically upon the ability of the hashing algorithm to distribute the allowable symbol space (or a subspace thereof, selected with heavy bias) more or less uniformly among the HN available slots in the hash entry table. The method designated "algorithm 1" in HASP is a form of the widely used "division" hashing technique, and was selected on the basis of favorable experience as reported in Reference 2 (the "drawback" reported there does not apply when collisions are resolved by direct chaining). However, there can be no guarantee that this algorithm will perform well for every set of codes nor for every set of symbols, so logical "room" has been left for other algorithms.

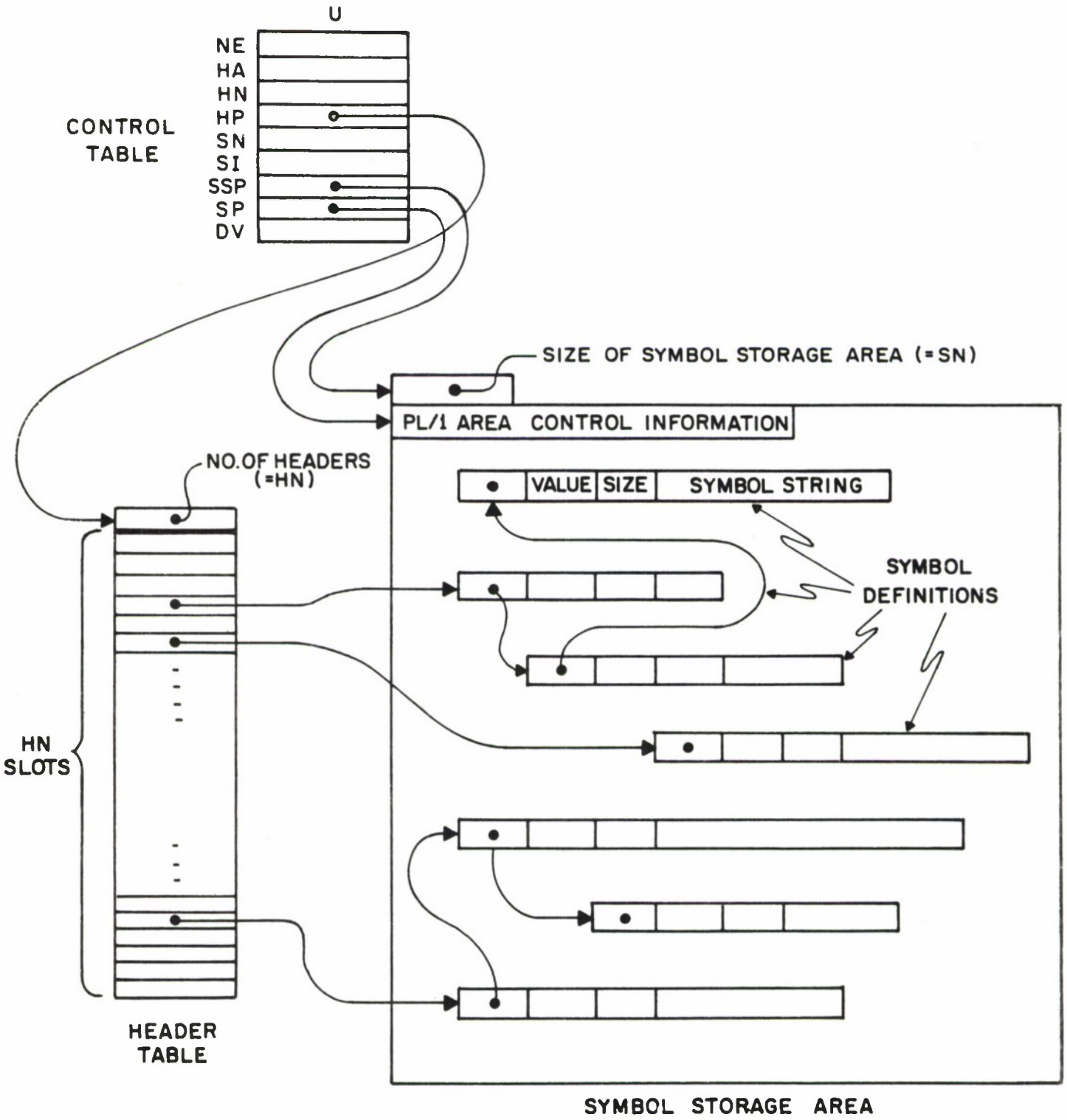
Algorithm 1 works as follows:

(1) Pad the string on the right with up to 3 "FF_{16}" characters to make up an integral multiple of 4 characters.

(2) Sum up the 32-bit integers formed by successive groups of 4 characters, using 2's complement arithmetic and discarding carries from the leftmost bit.

(3) Add the complement of the leftmost bit to the whole sum, again neglecting carryout.

(4) Set the leftmost bit to 0 and find the modulus of the resulting 31-bit positive integer with respect to HN. Add 1. The result is the hash entry number.



IB-28,289

Figure 1. HASP TABLE RELATIONSHIPS

Hash Entry Table

The hash entry (header) table is a set of HN offset pointers, each of which designates a list of current symbols all hashing to that particular entry. Its size is 4 x HN bytes. Normally, HN is set to approximately the average number of entries (NE) expected to reside in the table at one time, but this is not essential if one wishes to choose another point in the space-time tradeoff curve. If the hashing algorithm is assumed to select slots according to a uniform distribution, the average number of probes required to find an item known to be in the table is

$$\overline{P} = \frac{(NE)-1}{2(HN)} + 1.$$

Symbol Storage Area

The symbol storage area is a contiguous area of core, distinct from the header table, containing the symbols and their values. Each entry requires 12 bytes plus the length of the symbol string (rounded up, in effect, to the nearest multiple of 4 bytes). When an entry is deleted, the space usually becomes a "hole" suitable only for entries of similar or shorter length. This characteristic will adversely affect the effective density of a table containing variable-length symbols after a series of additions and deletions.

Accessing the Header Table and Symbol Storage

For special purposes, such as saving or restoring of a table via I/O, or when using HPLU or HALL (q.v.), it is necessary to "get at" the tables directly. Depending on the purpose, one or more of the following declarations will permit such access:

```
DCL  1  HEDS BASED (U. HP),                                    [1]

        2  HEDSIZ FIXED BIN,
        2  HEDRS (U.HN REFER (HEDSIZ))
             OFFSET(SPACE);

DCL  SPACE AREA BASED (U.SP);                                  [2]

DCL  1  NSPS BASED (U.SSP),                                    [3]

        2  NSPSIZ FIXED BIN,
        2  NSPACE AREA (U.SN REFER (NSPSIZ));
```

```
DCL  1  SYMDEF BASED (P),                                    [4]

          2  CHN OFFSET (SPACE),
          2  IVALUE FIXED BIN(31),
          2  CSIZE FIXED BIN,
          2  CSTR CHAR (CN REFER(CSIZE)),

        P PTR, CN FIXED BIN;
```

Note that none of the first three declarations will be allowed by the compiler if U is BASED (although they are permissable if U is a parameter).

HEDS is the header table. SPACE is the symbol storage area, actually part of the structure NSPS (SPACE and NSPACE are equivalent). Each entry in the table has the form of SYMDEF.

Declarations (1) and (4), as written, require the presence of declaration (2). If the offsets are not actually to be used, however, it is often more convenient to replace OFFSET (SPACE) in (1) or (4) with BIT(32) or something similar.

GENERAL PROPERTIES OF THE STORE AND RETRIEVE FUNCTIONS

While a table is in use (after a HASP call and prior to a HRLSE call), entries are made and referenced by calls to other HASP entry points. In every case, the "universe" U is passed as an argument to the package. If the particular table entry must be identified by symbol, STRING - a fixed or varying length character string of arbitrary size - is passed for this purpose. Where appropriate, the "value" of the symbol is passed into or returned from the HASP package as IVAL. Within HASP, IVAL is considered to be BIN FIXED(31) - a widely used data type in itself, as a table subscript, or as a key to a REGIONAL (1) file. However, using the PL/1 UNSPEC function, the user can make the value anything he wishes that will fit in 32 bits, including

POINTER,

OFFSET,

CHAR(4),

BIN FLOAT(21), and

BIT(32).

In several cases, provision is made for logical (BIT(1)) error indicators to be returned by HASP. EX denotes an error because a symbol, expected to be a new entry from the form of the call, was found to exist (to have a previous definition). NEX denotes the complementary situation.

Every entry-specific store or retrieve call to HASP will cause two EXTERNAL items, HASHEN and HCLASH, both BIN FIXED (31), to be set. These are, respectively, the hash entry number (the subscript to the header table) and the number of probes made into the collision chain before the entry (or the end of the list) was found - i.e., HCLASH $\geq$ 0. These items are useful primarily for instrumentation.

ADDING AND MODIFYING ENTRIES

HSTOW - Add or Modify

The statement

```
CALL HSTOW (U, STRING, IVAL);
```

will cause IVAL to become the value of STRING in U - creating a new entry in the table or replacing a previously defined value.

HSNEW - Add Only

The statement

```
CALL HSNEW (U, STRING, IVAL, EX);
```

will cause a new entry for STRING, with the value IVAL, to be made in U - provided an entry for STRING is not already present. In this case EX is returned reset.* Otherwise, EX is returned set and the table is not affected.

HSOLD - Modify Only

The statement

```
CALL HSOLD (U, STRING, IVAL, NEX);
```

will cause IVAL to become the new value for STRING in U, provided an entry for STRING already exists in the table. In this case, NEX is returned reset. Otherwise, NEX is returned set and the table is not affected.

*"Reset" means '0'B (false); "set" means '1'B (true).

DELETING ENTRIES

HDROP - Delete or Ignore

The statement

```
CALL HDROP (U, STRING);
```

will cause the entry for the symbol STRING, if any, to be deleted from the table. If no such entry exists, there is no effect.

HDOLD - Delete Only

The statement

```
CALL HDOLD (U, STRING, NEX);
```

will cause the entry for the symbol STRING to be deleted from the table, provided such an entry exists. In this case, NEX is returned set and the table is not affected.

HCLEAR - Grand Delete

The statement

```
CALL HCLEAR (U);
```

will cause all entries to be deleted. The table is left in an initialized state although the symbol storage area size (SN) may now be greater than when the table was originally initialized. Deletion of all entries is not equivalent to deletion (release) of the table itself, recovering the space (this is discussed in a later section).

RETRIEVING ENTRIES

IV-Value Function

The function reference

```
IV (U, STRING)
```

will yield the current value of STRING in U, if defined, or U.DV otherwise. IV is FIXED BIN (31).

IVLU - Value Look-Up Subroutine

The statement

```
CALL IVLU (U, STRING, IVAL, NEX);
```

will cause IVAL to be set to the value of STRING in U, if such an entry appears in the table. In this case, NEX is returned reset. Otherwise, IVAL is returned set to U.DV and NEX set.

HPLU - Pointer Look-Up Function

The function reference

```
HPLU (U, STRING)
```

will yield a pointer to the symbol definition for STRING in U if such exists, the pointer NULL otherwise.

This pointer is the base for SYMDEF (see "Accessing the Header Table and Symbol Storage" above) and so permits direct reference to the elements thereof. The value associated with STRING, the element SYMDEF.IVALUE, may also be modified directly. The other elements cannot simply be modified; additional provisions must be made to preserve the structural integrity of the table.

HALL - Grand Retrieve

The call

```
CALL HALL (U, PA);
```

where PA is an array of pointers, will retrieve all current entries in the form of pointers to the SYMDEF structures (compare HPLU above). PA may be of any storage class and should be dimensioned at least equal to NE at the time of call.

If, due to contamination, the number of entries found by HALL is not equal to NE, an ERROR condition is signalled. No more than NE pointers will be stored in PA in any case.

The order of entries returned will be arbitrary from the user's viewpoint (major sort by hash entry, minor sort by position in clash chain). Of course, they may be rearranged in any order to suit the user's needs (e.g., an alphabetic list of symbols).

DELETING THE TABLE (HRLSE)

The statement

```
CALL HRLSE (U);
```

will cause the table U to be deleted and the space occupied by the headers and symbol storage to be released. The table is left in an uninitialized state; HASP must be called before it can be used again. Note that U should be initialized at the time HRLSE is called.

SECTION III

PROGRAM AVAILABILITY

HASP presently exists in "NCAL" load module form as member HASP in AALIB on disk pack DP5010. If AALIB is concatenated with SYS1.PL1LIB in one's SYSLIB DD statement, references to HASP are automatically resolved and the control sections automatically included.

To alleviate coding tedium, a data set containing PL/1 text, suitable for % INCLUDE reference, has been placed on disk pack DP5010 as member HASPINC of PDS AAINC. This text consists of preprocessor statements defining three preprocessor symbols

HASP_ES#,

HASP_S#, and

HASP_P#.

HASP_S# is a string variable whose value is the "U" structure declaration save for the "DCL 1 U," and the terminal semicolon. Hence, a table can be declared by coding only

```
DCL 1 MYTABLE, HASP_S#;
```

HASP_ES# yields a string which, when coded in the parameter list of an ENTRY declaration, designates that parameter as a HASP structure, e.g.,

```
DCL MYSUBR ENTRY (HASP_ES#, BIN FLOAT (21));
```

HASP_P# yields a complete declaration for all HASP entries, wherever the "statement"

```
HASP_P#;
```

is coded.

If HASPINC is used, INCB - also in AAINC - must also be included.

REFERENCES

1. Morris, Robert, "Scatter Storage Techniques", Communications of the ACM, Vol. 11, No. 1

2. Maurer, W. D., "An Improved Hash Code for Scatter Storage" Communications of the ACM, Vol. 11, No. 1 (January 1968), 35-38

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)* The MITRE Corporation Bedford, Massachusetts	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

HASP: A PL/1 Hash Storage Package

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

N/A

5. AUTHOR(S) *(First name, middle initial, last name)*

Joseph E. Sullivan

6. REPORT DATE 30 July 1969	7a. TOTAL NO. OF PAGES 17	7b. NO. OF REFS 2
8a. CONTRACT OR GRANT NO. F19(628)-68-C-0365 b. PROJECT NO. 512C c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) MTR-927 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES N/A	12. SPONSORING MILITARY ACTIVITY Director of Planning and Technology, Electronic Systems Division, Air Force Systems Command, USAF, L.G. Hanscom Field, Bedford, Massachusetts.

13. ABSTRACT

A "generalized" PL/1 program is described which permits the assignment of "values" to a set of symbols, and the retrieval of the value assigned to a given symbol. That is, functions of character strings may be defined and evaluated. The method used is hash entry with chaining.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Hash Addressing						
Hash Code						
Hash Table						
Scatter Storage						
Searching						
File Searching						
File Addressing						
Storage Layout						