51

219-6641A

RECOGNITION OF IDENTICAL STUBS

IN A

DECISION TABLE PROCESSOR

By

CHI-DONG LU

B.S., National Taiwan University, 1968

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1973

Approved by:

[illegible]

Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A decision table is simply a tabular form for expressing conditional logic. It shows the relationship between conditions, the values of those conditions and the associated actions.

The use of decision tables in computer source programs and the development of processors for converting these decision tables to standard source programs has occurred over the past few years. For example, the Census Bureau and Working Group 2 of the Special Interest Group for Programming Language (of the Los Angeles Chapter of ACM) have each independently developed a preprocessor to convert decision tables written in COBOL to standard COBOL statements and paragraphs. These can be converted to a computer object program by a COBOL compiler.

Recently, Smith and Conrow are developing the DECTAL processor to convert decision tables written in PL/I to standard PL/I statements, which can be translated to a computer object program by a PL/I compiler.

Due to the complex logic in decision making, a group of linked decision tables are used for segmenting the logic shown in decision table form. A key feature in DECTAL is the ability to reference condition or action stubs anywhere in a block of decision tables from any other point in the block. Currently it is the programmer's responsibility to supply cross references if he wishes to effect the economy in compile time and execution module size which a cross reference implies. Automatic recognition of identical stubs and provision of the cross references while still providing the desired economies.

The program developed in this work is written to be incorporated into the DECTAL processor to do the automatic recognition of identical stubs. The cross-reference table already existing in DECTAL can be modified for identical stubs recognized as the result of this program.

II. THE STRUCTURE OF A DECISION TABLE

The general basic structure of a decision table is shown in Figure 1. There are four sections or quadrants usually separated by double lines but often in other ways, e.g. single line, thick line. The sections set our respectively: the full set of conditions applicable (condition stubs), the full set of actions applicable (action stubs), the different combinations of those conditions(condition entries) and the corresponding combinations of actions (action entries). Each combination of conditions and associated actions forms a decision rule set out in parallel to all the other decision rules.

Within this general structure, several different types of decision tables can be defined. One classification is based on whether or not the rules are set out in columns or rows; the former is termed "vertical rule" and the latter "horizontal rule".

A more important difference lies, however, in the extent to which conditions and actions are defined in the stub. Where conditions and actions are wholly specified in the stub (as illustrated in Figure 2), the table is termed "limited entry". Entries are limited to noting the status of particular conditions and actions in a specific rule.

In those cases where conditions and actions are generally identified in the stub, with specific values shown in the entries, the table is termed "extended entry". Figure 2 illustrates horizontal rule as well as extended entry.

The last important difference lies in the way in which the conditions are linked. The conditions can be linked by the connector AND, OR or MIXED AND/OR. In Figure 2 and 3 they are linked by connector AND.

DECTAL handles extended entry decision tables by converting them to limited entry decision tables, so the program described here concerns itself solely with recognition of identical stubs in a limited entry decision table.

III. CROSS REFERENCES

It would be unrealistic to define logic in all cases in a simple table, so a group of tables is desired to express the required logic. A hypothetical example of the linkage between tables in a flowchart is illustrated in Figure 4.

As a result of using a number of tables for segmenting the logic, a table and a stub need to be identified. The symbol Txx is recognized to denote the heading of a table (e.g. T40 means table forty). In the same manner the symbol A(or C)xx designates a specific Action (or Condition) stub in a certain table in the DECTAL processor. The combination of these two symbols can completely identify the position of a stub in the set of tables. Using these identifying symbols, the DECTAL processor constructs a cross-reference table for all the stubs with the Form *Txx (A|C)xx to convey programmer-supplied referencing of the identical stubs among the tables.

This program groups all the identical stubs together and could be incorporated into the DECTAL processor to modify the cross reference table for identical stubs.

Condition Stubs	Conditions Entries
Action Stubs	Action Entries

Figure 1. Basic structure of a decision table

	RULES			
	1	2	3	4
Credit Satisfactory ?	Y	N	N	N
Prompt payer	-	Y	N	N
Special Clearance Obtained ?	-	-	Y	N
Accept order	x	x	x	
Return order to sales				x

Figure 2. A limited entry vertical rule table

		CONDITIONS			ACTION
		Product	Customer	Order value	Discount
	1	Battery	Retailer	$50	15%
R	2	Battery	Wholesaler	$50-$200	22%
U	3	Battery	Contract	$50-$200	27%
L	4	Battery	Agency	$50-$200	25%
E	5	Tires	Retailer	$50	10%
S	6	Tires	Wholesaler	$50-$200	29%

Figure 3. An extended entry horizontal rule table

IV. THE APPROACH OF THE PROGRAM

The DECTAL processor handles the decision tables written in the PL/I language. In general, the necessity for inserting a blank between terms in PL/I statements implies the ability to use any number of blanks at that point; and the location of the comment (/* ---*/) is not seriously restricted in the PL/I language; it can appear in front of, following, or even in the middle of a statement.

The possibility that two identical stubs would not be recognized as identical because of extra blanks or of a comment included in one or both must be removed. Reduction of stubs to a standard, deblanked, decommented form, will permit simple character string comparison to detect identical stubs. Such comparison is a very low-level test for identity; stubs like 'IF I=1' and 'IF 1=I', which are identical in effect though different as character strings, can never be recognized by the methods of the program developed here.

The literals in PL/I statements mean the exact form in the single quotation marks must be taken, so that the literals cannot be deblanked.

After the standard form of the stub is set up, the grouping of the stubs is introudced here. We apply the linked allocation list method to link all the stubs in the tables, although it is easy to insert an item into the midst of a list, but it takes a long time to search for the appropriate position for an item.

Ten lists are used instead of a single long list for lowering the search time to about one tenth of that for a single long list. Ideally, we would like to link all the equal length stubs in a list, but since

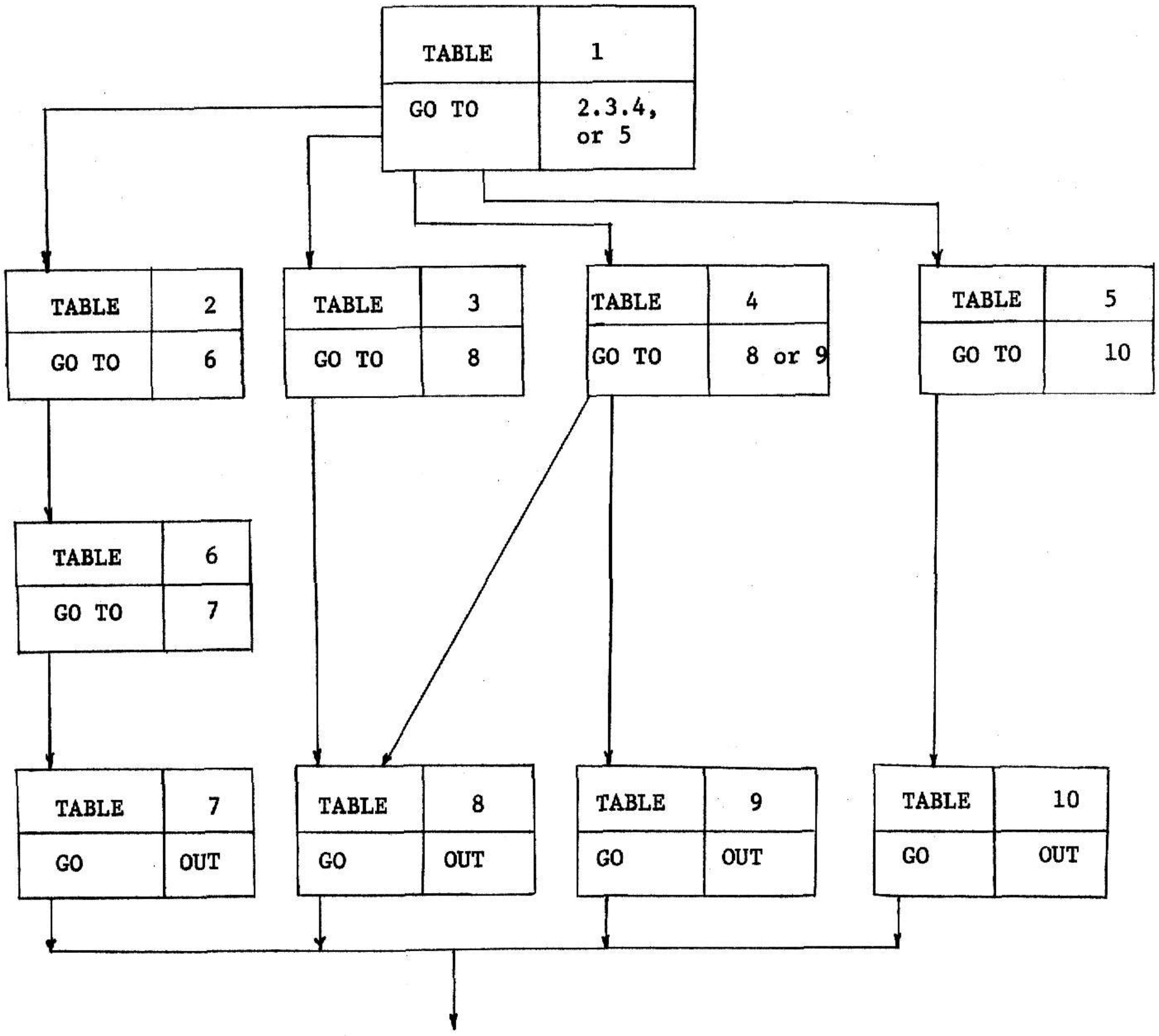


Figure 4. Table linkage flowchart

a limited number of lists can be used effectively in a program, in our case, the stubs are divided among 10 lists according to their length as shown in the following table:

l : stub length

Condition stubs					Action stubs				
1	2	3	4	5	6	7	8	9	10
$0 < l \leq 16$	$16 < l \leq 32$	$32 < l \leq 48$	$48 < l \leq 64$	$l > 64$	$0 < l \leq 16$	$16 < l \leq 32$	$32 < l \leq 48$	$48 < l \leq 64$	$l > 64$

The first five lists take care of the condition stubs, while the rest of lists handle the action stubs. All lists are contained in an area with size of 20,000 bytes.

Since each list needs a ground node to indicate its termination, a single blank stub node is used to be the ground node of each list. This is the most efficient way to set up each list in descending order because all stubs can be inserted in a pre-existing list without need to test for special cases.

The identical stub's identifier for each subsequent appearance of a stub is linked into the identical stub list attached from the first appearance stub in the first-in-first-out manner while the first appearance stub is linked in the stub list in descending order.

The space restrictions imposed by the already large size of the input section of DECTAL will force the use of a much smaller area by this program using ten areas each one containing one list instead of using ten lists in one area will permit saving some core memory by swapping the lists in and out of core. When ten areas are used, extra execution time for the swapping of areas will be paid.

V. INPUT/OUTPUT

The input for the program has two parts, which will be transfered from the DECTAL processor when the program is incorporated into DECTAL. One part is the stub identification field and the other is the stub statement. An input is handled in each cycle, that is the next input data is read right after an input is completely handled, until the '*EN' in the stub identification field (IDS) is read, the program will transfer its control back to the DECTAL processor.

The program handles unlimited input, but each stub cannot exceed 710 characters, because 710 characters is the largest limited entry stub which can be generated from an extended entry statement. DECTAL can handle indefinitely long limited entry stubs. The main excuse for a limit of 710 characters is that significantly long passages of identical code will be easy for the user to cross reference for himself. Presumably only short identical stubs will be reused with sufficient frequence that automatic recognition is practical and necessary.

Since DECTAL processes stubs one at a time and puts entries in the directory vector shortly after each stub is processed, it seems better to let this program be a subroutine which is called with the stubs and identifier fields as arguments and returns either the same identifier field (if the stub is a new one) or the identifier field of the stub at its earlier occurence (if the stub is a repeat). Then DECTAL would use the returned identifier field in its dictionary look-up process just as it currently uses the programmer-supplied cross reference identifier field.

VI. PROGRAM MATERIALS AVAILABLE

1. Documentation : 22 pages.
2. Flowchart : 1 chart.
3. Listing : 3 pages.
4. Source deck : 122 cards.

VII. LIMITATIONS

1. Machine configuration : IBM system 360 model 50.
2. Code : PL/I(F) version 5.3B.
3. Others : NEATER2 (A PL/I Source Statement Reformatter) is used.

VIII. ALGORITHM

1. Initialize ten head pointers and create a ground level stub node for each pointer.
2. Read input data.
3. If '*EN' is read in the stub identification field of the input then go to step 10.
4. Remove comments in the stub.
5. Deblank the stub except for literals inside the single quotation marks.
6. Allocate the stub node in the area.
7. Search for the appropriate linked list and position.
8. Link the stub node into the lsit.
9. Go to step 2.
10. Print out the cross reference table.
11. End the program.

IX. *IDENTIFIERS*

IDS : 6-character string holds the stub identifier of the current data item.

STUB : varying character string holds the stub statement in the current data item up to 710 characters.

STUBBUF : 1420-character string, converts the varying string STUB into a fixed character string and performs all the manipulation of the STUB.

STUBARY : an array has 710 elements; in which each element is 1-character string, defined on STUBBUF, such that every character can be accessed individually.

BS710 : 710-character string, pointed to by P, overlays on STUBBUF at an arbitrary position of STUBBUF.

HEADPT : a head pointer array, has 10 elements, each element points to a linked list.

Q1, Q2, Q : intermediate pointers.

STIDL : 1-character string defined on the 4th character of IDS to tell if a stub is a *condition* stub.

A : an area with the size of 20,000 bytes.

DAFILE : a based major structure, whose allocated address is pointed to by DAPT, used as a stub node shown in Figure 5.

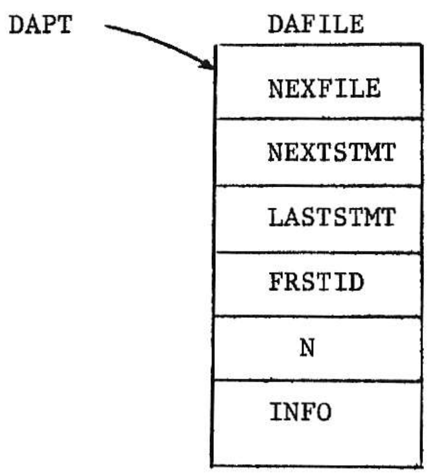


Figure 5. The structure of a stub node.

NEXFILE : link pointer points to the next stub node.

NEXTSTMT : head pointer points to the first node of any EQFILE list of stubs identical to this one.

LASTSTMT : end pointer points to the last node of any EQFILE list of stub identical to this one.

FRSTID : fixed length character string in DAFILE holds the stub identifier for the first appearance of a stub.

M, N : indicates the length of the stub.

INFO : varying character string in DAFILE, copies the stub statement.

EQFILE : major based structure, which allocated address is pointed by EQPT, used as an identical stub's identifier node shown in Figure 6.

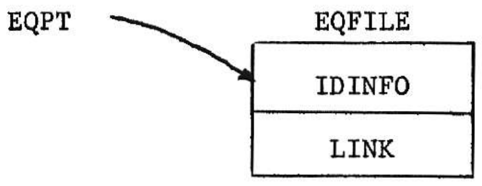
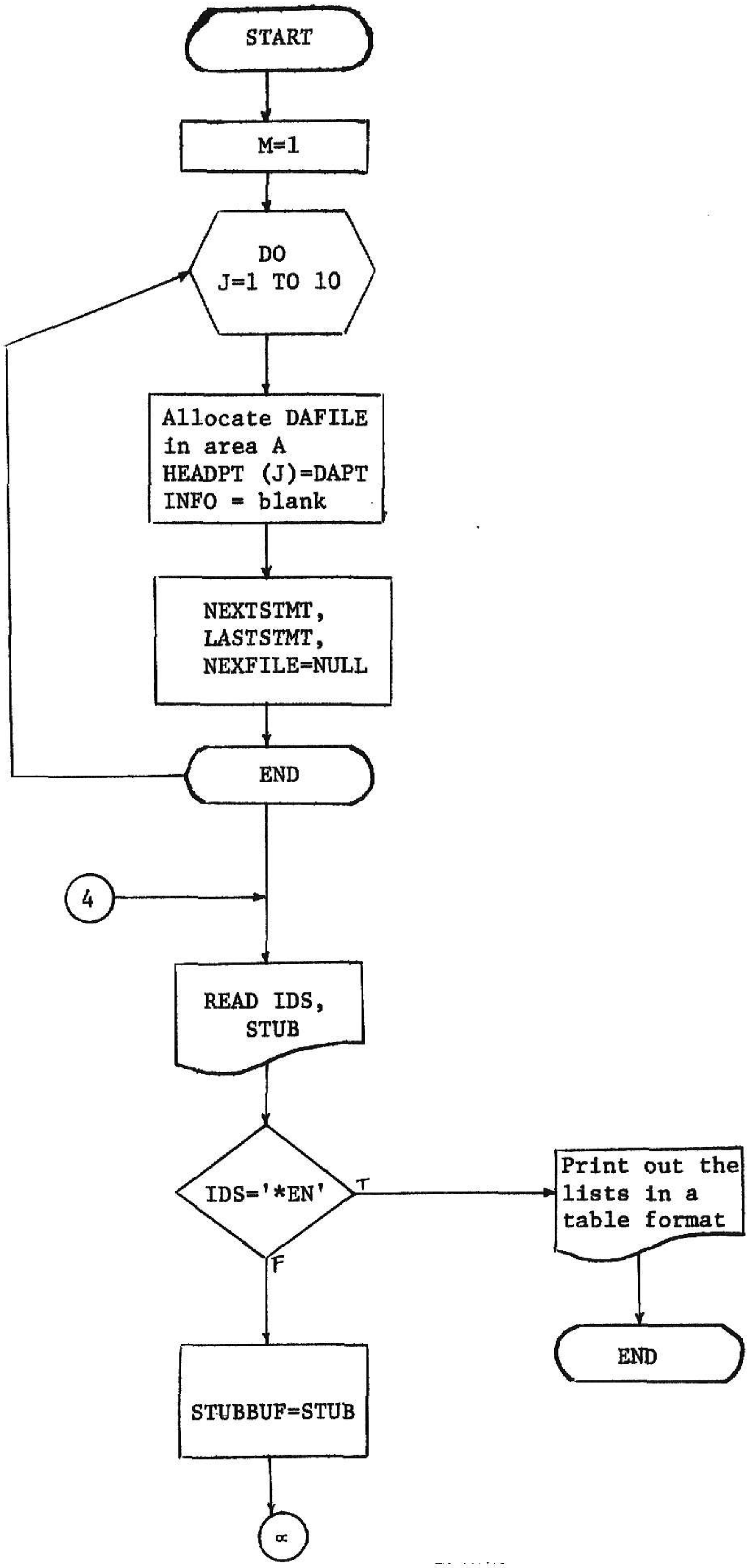


Figure 6. The structure of an identical stub's identifier node.

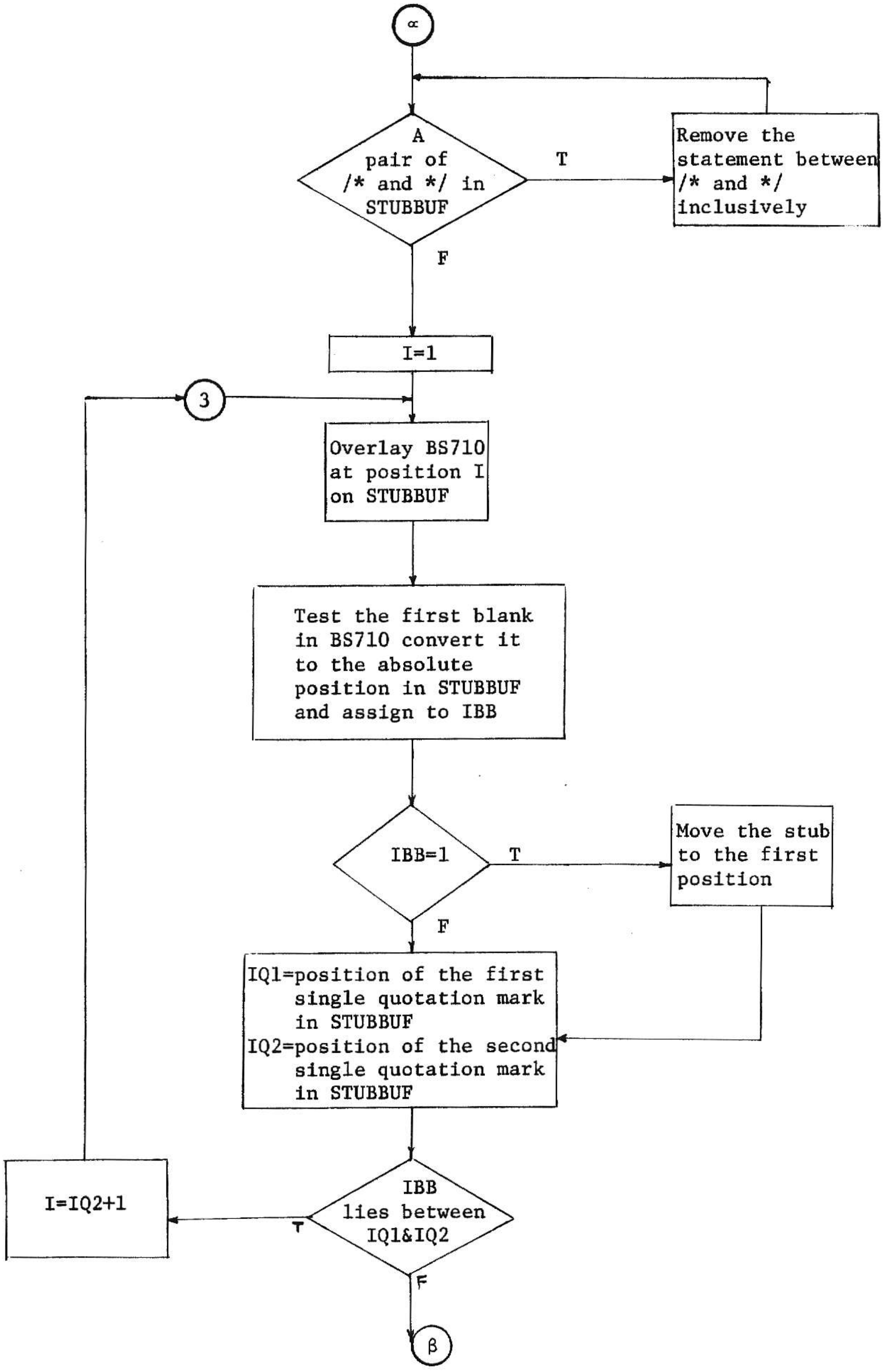
IDINFO : holds the identical stub's identifier for each
subsequent appearance of a stub.

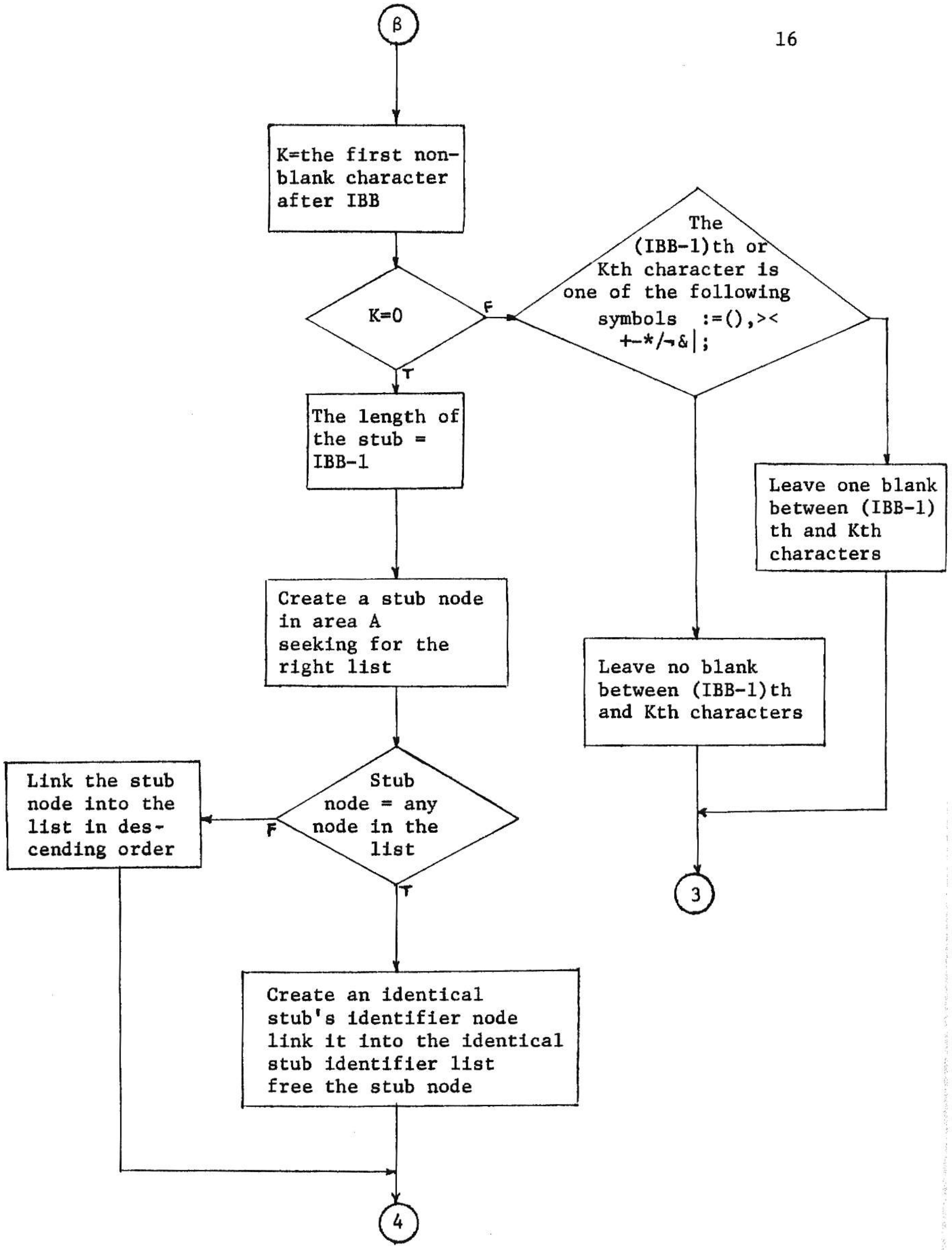
LINK : a pointer points to the next EQFILE node.

X. FLOWCHART



START
M=1
DO
J=1 TO 10
Allocate DAFILE in area A
HEADPT (J)=DAPT
INFO = blank
NEXTSTMT,
LASTSTMT,
NEXFILE=NULL
END
4
READ IDS,
STUB
IDS='*EN'
T
F
Print out the lists in a table format
END
STUBBUF=STUB
∝


∝
A pair of /* and */ in STUBBUF
T
Remove the statement between /* and */ inclusively
F
I=1
3
Overlay BS710 at position I on STUBBUF
Test the first blank in BS710 convert it to the absolute position in STUBBUF and assign to IBB
IBB=1
T
Move the stub to the first position
F
IQ1=position of the first single quotation mark in STUBBUF
IQ2=position of the second single quotation mark in STUBBUF
IBB lies between IQ1&IQ2
T
I=IQ2+1
F
β


β
K=the first non-blank character after IBB
K=0
F
T
The (IBB-1)th or Kth character is one of the following symbols :=(),>< +-*/¬&|;
The length of the stub = IBB-1
Leave one blank between (IBB-1) th and Kth characters
Create a stub node in area A seeking for the right list
Leave no blank between (IBB-1)th and Kth characters
Link the stub node into the list in descending order
F
Stub node = any node in the list
T
3
Create an identical stub's identifier node link it into the identical stub identifier list free the stub node
4

XI. EXPLANATION

A. In the initialization block we set up the ground stub node in order to indicate the last node of the list, such that we know when to stop processing the list. There are ten lists of this kind in area A. The snapshot of a list is shown in Figure 7.

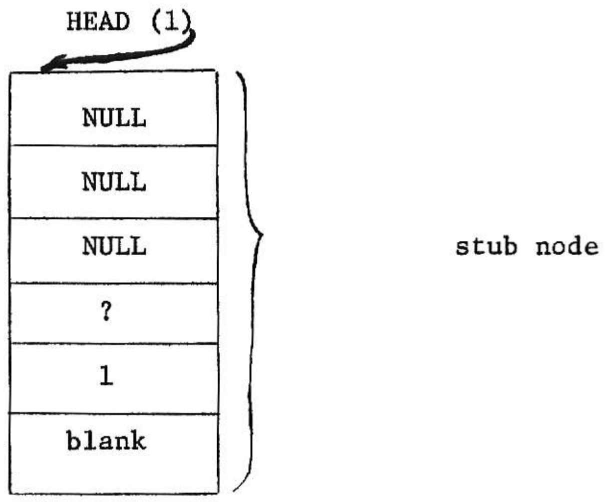


Figure 7. The snapshot of an initialized node list.

Each node can have its own list known as the identical stub's identifier list which contains all the identical stub's identifiers as will be explained later.

B. After reading in an input data we have to deblank the stub into a standard form. First of all we remove all the comments in the stub then progress to the deblanking process as shown in following.

Assume that after removing all comments the stub looks like

```
                  PUTbbLISTbb('bTEST');
Position :        1234567891111111111122
                           012345678901
```

The deblanking process will do the following steps :

1. I=1 indicates the first character of the stub.
2. IQ1 and IQ2 indicate the positions of the first single quotations after I.
3. Find the first blank position after the Ith character and store in IBB.
4. If IBB falls between IQ1 and IQ2 then let I point to the character next to IQ2 (i.e. I=IQ2+1) and go back to step 2.
5. K indicates the first non-blank position after the IBBth character.
6. If there is no more non-blank character then terminate the deblanking process.
7. If (IBB-1)th or Kth character is one of the following symbols :

   :=(),><+-*/¬&|;

   then remove all blanks between IBB-1 and K; otherwise leave one blank between them.
8. Let I point to the first non-blank position at or after IBB and go back to step 2.

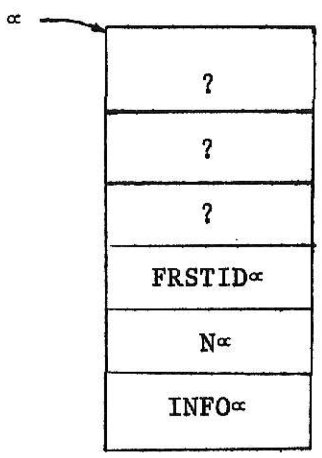
The deblanking of the stub can be described as follows

By step

```
   1          I=1

   2          IQ1=13, IQ2=19

   3          IBB=4

   4          IBB does not fall between IQ1 and IQ2

   5          K=6

   6          There is a non-blank character indicated

              by K

   7          Stub = PUTbLISTbb('bTEST');
                     12345678911111111112
                              01234567890

   8          I=5

   2          IQ1=12, IQ2=18

   3          IBB=9

   4          IBB does not fall between IQ1 and IQ2

   5          K=11

   6          There is a non-blank character indicated

              by K

   7          Stub = PUTbLIST('bTEST');
                     123456789111111111
                              012345678

   8          I=9

   2          IQ1=0, IQ2=0

   3          IBB=19

   4          IBB does not fall between IQ1 and IQ2

   5          K=0

   6          K indicates that there are no more non-blank

              characters so the process is completed
```

C. After we have the standard form of a stub, we have to create a stub node for the stub then search for its appropriate position (in descending order) in its list. If the stub is new, the node is inserted in the correct position of the list of stubs. If there is a stub node containing an identical stub statement in the list then we create an identical stub's identifier node linked into the list attached from stub node and free the created stub node. The above algorithm is described as the following Figures :

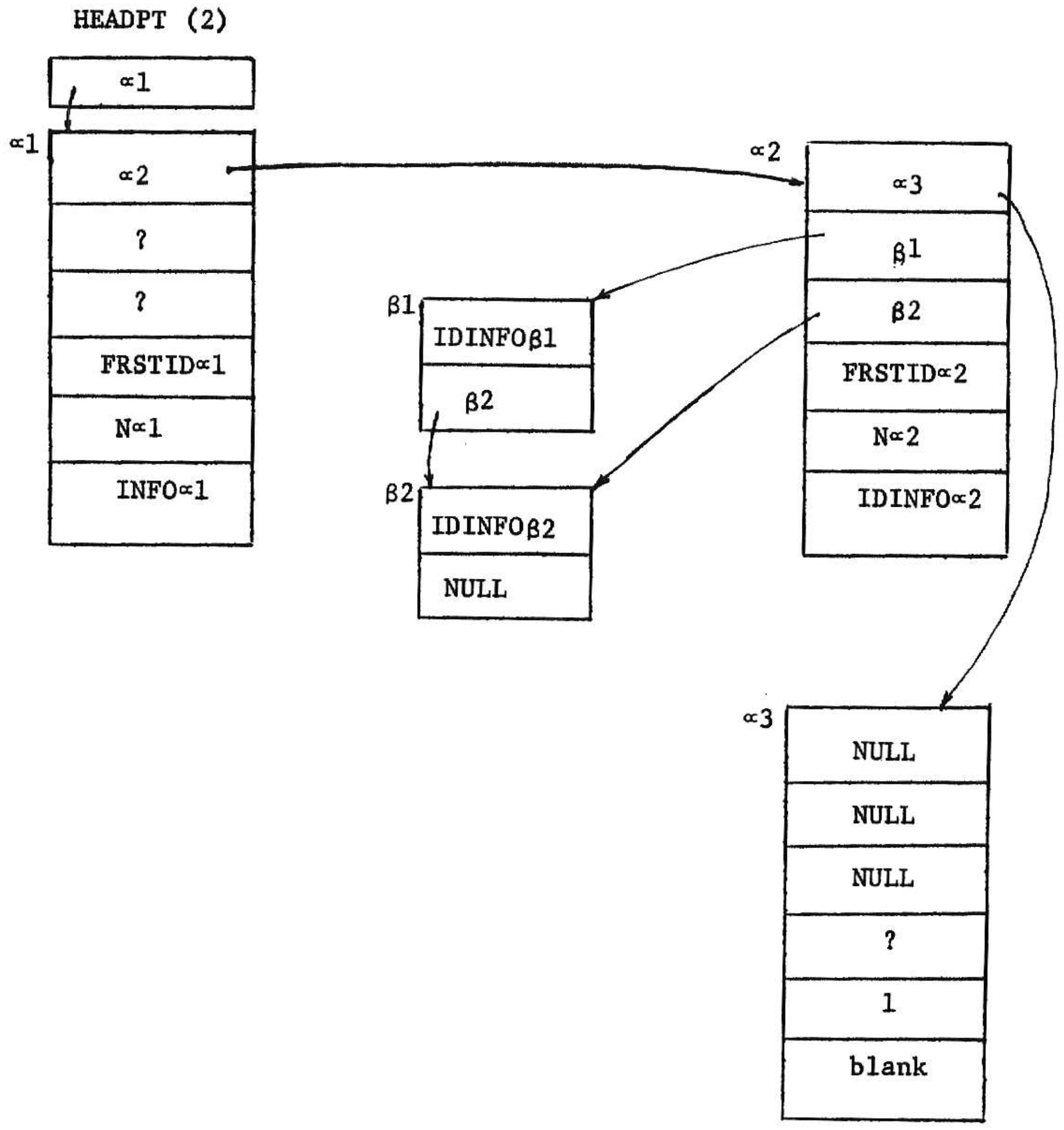
Assume the stub node known as



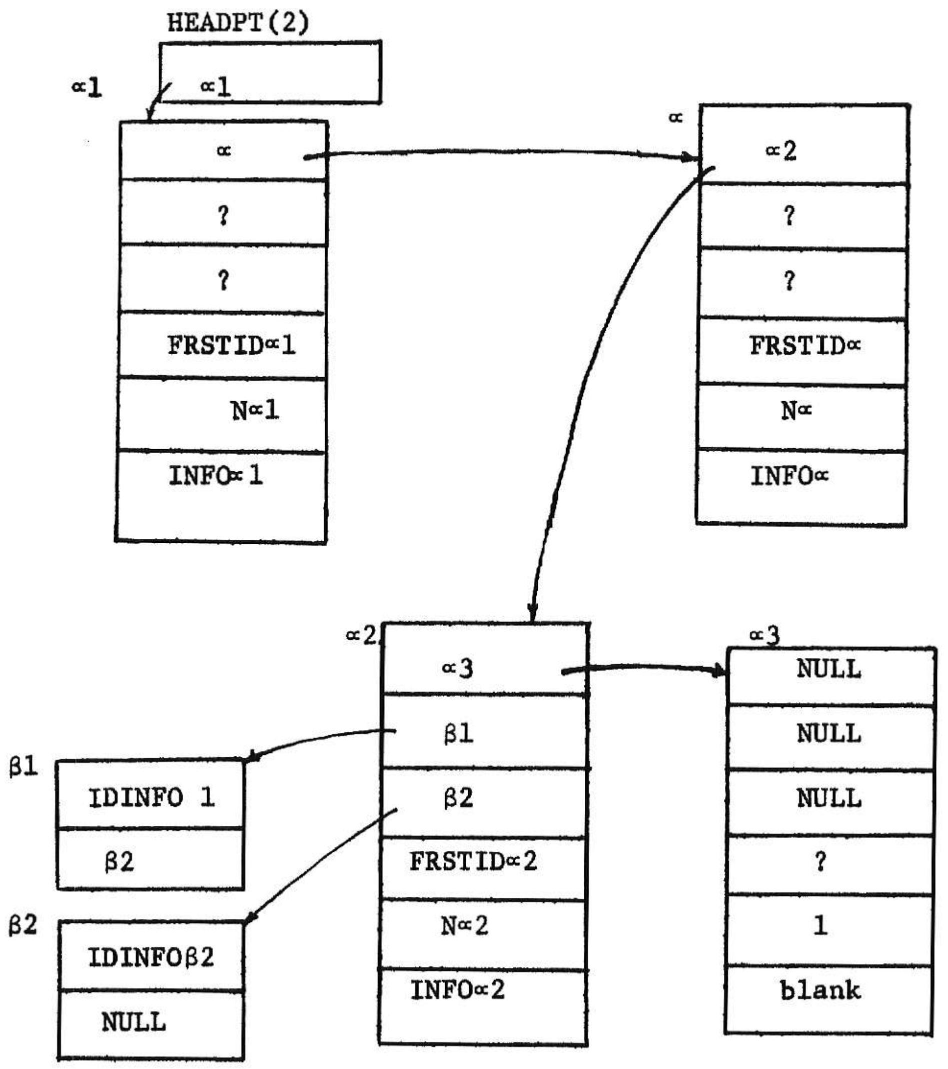
is assigned to the list pointed by HEADPT (2)

THIS BOOK
CONTAINS
NUMEROUS PAGES
WITH DIAGRAMS
THAT ARE CROOKED
COMPAIRED TO THE
REST OF THE
INFORMATION ON
THE PAGE.

THIS IS AS
RECEIVED FROM
CUSTOMER.

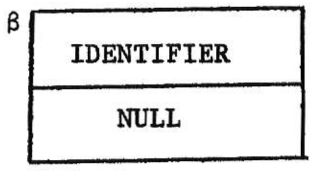

HEADPT (2)
∝1
∝1
∝2
?
?
FRSTID∝1
N∝1
INFO∝1
∝2
∝3
β1
β2
FRSTID∝2
N∝2
IDINFO∝2
β1
IDINFOβ1
β2
β2
IDINFOβ2
NULL
∝3
NULL
NULL
NULL
?
1
blank

If INFO∝1 > INFO∝ > INFO∝2 then the ∝-node is linked into the list as

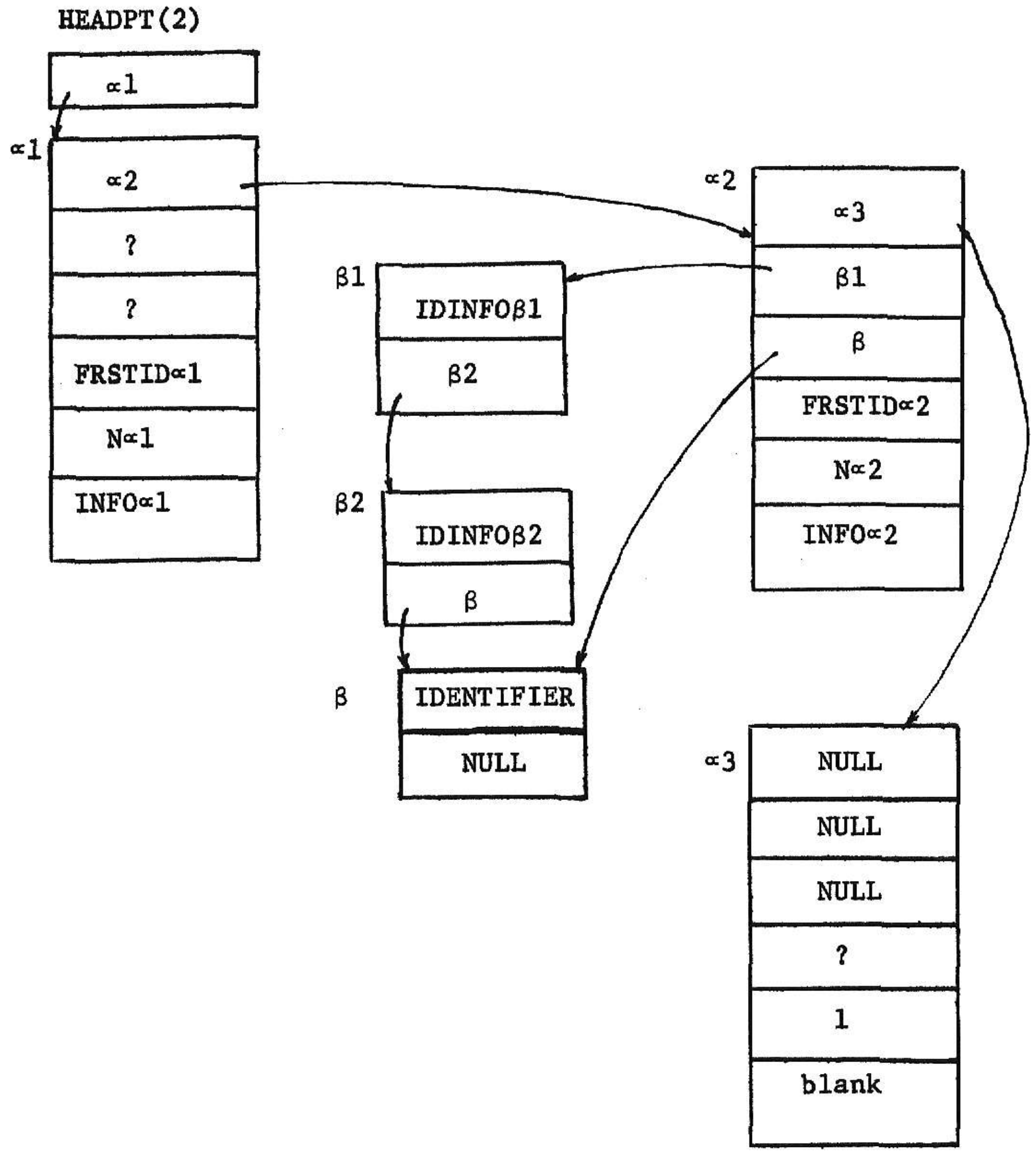


If INFO∝ = INFO∝2 then

create an identical stub's identifier node



Free the created stub node for ∝ and link the new β-node into the identical stub list attached from ∝2-node (in first-in-first-out manner) as



D. After handling all the input data there are ten stub node lists in the area A. According to these lists the cross reference table printout of DECTAL can be modified.

XII. CONCLUSION

The program does the automatic recognition of identical stubs in decision tables to be incorporated into the DECTAL processor, which converts decision tables written in PL/I to standard PL/I statements, for freeing its user from the necessity to supply the key feature cross references.

Ten lists are used to save one tenth of search time in a long list by using the linked allocation list method to insert an item into the midst of a list.

It takes 39.6 seconds execution time in slow core for the test run of this program on 48 stubs including the I/O process. If the program is incorporated into DECTAL, the I/O process can be eliminated to reduce the execution time.

There is an alternative method in using ten areas instead of using ten lists to save some core memory when the use of smaller area is forced by the large size of the input section of DECTAL.

XIII. PROGRAM LISTING

```
STUBREC:PROC OPTIONS(MAIN);                                                1
        DCL IDS CHAR(6),DCL IDS CHAR(6),STUB CHAR(710)VAR,STUBBUF CHAR(    1
        1420),STUBARY(710)CHAR(1)DEF STUBBUF,BS710 CHAR(710)BASED(P);      1
        DCL HEADPT(10)POINTER,(Q,Q1,Q2)POINTER,STIDL CHAR(1)DEF IDS POS    1
        (4),A AREA(20000),1 DAFILE BASED(DAPT),2 NEXFILE POINTER,2 NEXT    1
        STMT POINTER,2 LASTSTMT POINTER,2 FRSTID CHAR(6),2 N BINARY FIX    1
        ED,2 INFO CHAR(M REFER(N)),1 EQFILE BASED(EQPT),2 IDINFO CHAR(6    1
        ),2 LINK POINTER;                                                  1
/*INITIALIZE 10 HEAD POINTERS FOR THE LISTS*/                              1
        M=1;                                                               1
           DO J=1 TO 10;                                                   2
           ALLOCATE DAFILE IN(A);                                          2
           HEADPT(J)=DAPT;                                                 2
           INFO=' ';                                                       2
           NEXTSTMT,LASTSTMT,NEXFILE=NULL;                                 2
           END;                                                            2
INDATA: GET LIST(IDS,STUB);                                                1
           IF IDS='*EN' THEN                                               2
           GO TO TERMINT;                                                  2
        STUBBUF=STUB;                                                      1
/*REMOVE COMMENT STATEMENTS IN THE STUB*/                                  1
TEXTIN: IX=INDEX(STUBBUF,'/*');                                            1
        IY=INDEX(STUBBUF,'*/');                                            1
           IF IY¬=0 THEN                                                   2
              IF IX¬=0 THEN                                                3
                 DO;                                                       4
                 P=ADDR(STUBARY(IY+2));                                    4
                 SUBSTR(STUBBUF,IX)=BS710;                                 4
                 GOTO TEXTIN;                                              4
                 END;                                                      4
/*TEST FOR SINGLE QUOTES AND BLANK*/                                       1
        I=1;                                                               1
ABTRACT:P=ADDR(STUBARY(I));                                                1
        IBB=INDEX(BS710,' ')+I-1;                                          1
           IF IBB=1 THEN                                                   2
           BS710=SUBSTR(BS710,VERIFY(BS710,' '));                          2
        IQ1=INDEX(BS710,'''')+I-1;                                         1
        IQ2=INDEX(SUBSTR(STUBBUF,IQ1+1),'''')+IQ1;                         1
           IF IBB>IQ1 THEN                                                 2
              IF IBB<IQ2 THEN                                              3
                 DO;                                                       4
                 I=IQ2+1;                                                  4
                 GOTO ABTRACT;                                             4
                 END;                                                      4
/*DEBLANKING*/                                                             1
        J=VERIFY(STUBARY(IBB-1),':=(),><+-*/¬&|;');                        1
        P=ADDR(STUBARY(IBB+1));                                            1
        K=VERIFY(BS710,' ');                                               1
           IF K>0 THEN                                                     2
              DO;                                                          3
              I=K+IBB;                                                     3
                 IF J=1 THEN                                               4
                 J=VERIFY(STUBARY(I),':=(),><+-*/¬&|;');                   4
              P=ADDR(STUBARY(I));                                          3
              I=IBB+J;                                                     3
              SUBSTR(STUBBUF,I)=BS710;                                     3
              GOTO ABTRACT;                                                3
              END;                                                         3
        M=IBB-1;                                                        1
        J=CEIL(M/16);                                                   1
        J=MIN(J,5);                                                     1
        ALLOCATE DAFILE IN(A);                                          1
           IF STIDL¬='C' THEN                                           2
           J=J+5;                                                       2
        Q1,Q=HEADPT(J);                                                 1
        FRSTID=IDS;                                                     1
        DAPT->INFO=STUBBUF;                                             1
/*SEARCH FOR THE APPROPRIATE POSITION*/                                 2
SEARCH:    IF Q->INFO>DAPT->INFO THEN                                   2
              DO;                                                       3
              Q1=Q;                                                     3
              Q=Q->NEXFILE;                                             3
              GO TO SEARCH;                                             3
              END;                                                      3
/*LINK INTO THE LIST*/                                                  2
           IF Q->INFO<DAPT->INFO THEN                                   2
              DO;                                                       3
              DAPT->NEXFILE=Q;                                          3
                 IF Q=Q1 THEN                                           4
                 HEADPT(J)=DAPT;                                        4
                 ELSE                                                   4
                 Q1->NEXFILE=DAPT;                                      4
              DAPT->LASTSTMT,DAPT->NEXTSTMT=NULL;                       3
              GO TO INDATA;                                             3
              END;                                                      3
        ALLOCATE EQFILE IN(A);                                          1
        FREE DAPT->DAFILE IN(A);                                        1
        EQPT->LINK=NULL;                                                1
        EQPT->IDINFO=IDS;                                               1
           IF Q->LASTSTMT=NULL THEN                                     2
           Q->NEXTSTMT,Q->LASTSTMT=EQPT;                                2
           ELSE                                                         2
              DO;                                                       3
              Q2=Q->LASTSTMT;                                           3
              Q->LASTSTMT,Q2->LINK=EQPT;                                3
              END;                                                      3
        GO TO INDATA;                                                   1
/*PRINT OUT THE XREF TABLE*/                                            2
TERMINT:   DO K=1 TO 10;                                                2
           Q1=HEADPT(K);                                                2
LOOKY:     ICOUNT=1;                                                    2
              IF Q1->NEXFILE=NULL THEN                                  3
              GO TO TEMP;                                               3
           Q=Q1->NEXTSTMT;                                              2
LOOKX:        IF Q=NULL THEN                                            3
                 DO;                                                    4
                 Q1=Q1->NEXFILE;                                        4
                    IF ICOUNT=0 THEN                                    5
                    PUT EDIT('.')(A);                                   5
                 GO TO LOOKY;                                           4
                 END;                                                   4
              IF ICOUNT=1 THEN                                          3
                 DO;                                                    4
                 IDS=Q1->FRSTID;                                        4
                    IF STIDL¬=' ' THEN                                  5
                    PUT SKIP(2)EDIT(Q1->FRSTID,' --')(A(6),A(3));       5
                    ELSE                                                     5
                    PUT SKIP(2)EDIT(Q1->FRSTID,' -----')(A(3),A(6));         5
                 END;                                                        4
              ELSE                                                           3
              PUT EDIT(',')(A);                                              3
           IDS=Q->IDINFO;                                                    2
              IF STIDL=' ' THEN                                              3
              PUT EDIT(Q->IDINFO)(X(1),A(3));                                3
              ELSE                                                           3
              PUT EDIT(Q->IDINFO)(X(1),A(6));                                3
           ICOUNT=0;                                                         2
           Q=Q->LINK;                                                        2
           GO TO LOOKX;                                                      2
TEMP:      END;                                                              2
        END STUBREC;                                                         1
```

ACKNOWLEDGMENT

The author wishes to express his sincere thanks and appreciation to his major professor, Dr. Kenneth Conrow, for his suggestion of this topic and assistance during the preparation of this report.

The help of Dr. Kane S. Yee and Dr. Myron A. Calhoun in reviewing the manuscript is gratefully acknowledged.

REFERENCES

1. Smith, Ronald G. and Conrow, Kenneth. DECTAL : A DECISION TABLE ALGORITHM, Kansas State University Computing Center, Manhattan, Kansas, 1972.

2. London, Keith R. Decision Tables, Auerback Publishers Inc., Princeton, New Jersey, 1972.

3. The National Computing Center Ltd. Decision tables in data processing, Science Associates/International Inc., New York, 1970.

4. Knuth, Donald E. Fundamental Algorithms, The art of Computer Programming, volume 1, Addison Wesley Publishing Company Inc.

5. Pollack, Seymour V. and Sterling, Theodor D. A Guide to PL/I, Holt, Rinehart and Winston, Inc., New York, 1969.

6. Smith, Ronald G. and Conrow, Kenneth. NEATER2 : A PL/I source statement reformatter, Comm. ACM, 13, (11), Nov. 1970, pp. 669-675.

7. IBM. IBM system/360 Operating system PL/I(F) Language, Reference Manual, Form GC28-8201-3.

RECOGNITION OF IDENTICAL STUBS

IN A

DECISION TABLE PROCESSOR

By

CHI-DONG LU

B.S., National Taiwan University, 1968

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1973

A decision table is also called logic table, decision logic table, decision structure table and step.

The development of the DECTAL processor converts decision tables written in PL/I to PL/I statements.

In many cases where complex logic is involved, more than one decision table is used for segmenting the logic shown in decision table form.

The cross reference table construction in DECTAL needs to be modified for the identical stubs in the tables.

This program is written to incorporate into the DECTAL processor to do the automatic recognition of identical stubs.

The program is coded in PL/I language and run through IBM system 360 model 50.

Deblanking and linear allocation list technology are used in the program.