INCORPORATING TABLES INTO TEXT

Submitted by

Roger de Peiza

Project Report

August 1984

T A B L E O F C O N T E N T S

1.1 Introduction 3
1.2 Introduction to MRDS/FS 4
1.3 MRDSA 5
1.4 Running MRDSA on the IBM PC 7
1.5 The Relational Approach 9
1.6 MRDSA User Procedures 12
1.7 MRDSA System Relations 13
1.8 MRDSA System Procedures 17
1.9 MRDSA User and System Procedures used by "TABSINTEXT" 19

2.1 Formatting Tables with Text 29
2.2 The "INPUT" Relation, the "TABLES" Relation, and the Data to be referenced 30

3.1 The Formatting Phase 39
3.2 The Formatting Algorithm 49

4.1 The Program "TABSINTEXT" 63
4.2 Setting up the Input for "TABSINTEXT" 74

APPENDIX A

APPENDIX B

APPENDIX C

APPENDIX D

APPENDIX E

BIBLIOGRAPHY

```
 CCCCC   HH    HH   AAAAAAA   PPPPPPP   TTTTTTTT   EEEEEEEE   RRRRRRR
CC       HH    HH   AA    AA  PP    PP     TT      EE         RR    RR
CC       HHHHHHHH   AAAAAAA   PPPPPPP      TT      EEEEE      RRRRRRR
CC       HH    HH   AA    AA  PP           TT      EE         RR  RR
 CCCCC   HH    HH   AA    AA  PP           TT      EEEEEEEE   RR    RR


                     111
                    11111
                   111111
                     111
                     111
                     111
                     111
                 111111111111111
```

1.1 Introduction

1.2 Introduction to MRDS/FS

1.3 MRDSA

1.4 Running MRDSA on the IBM PC

1.5 The Relational Approach

1.6 MRDSA User Procedures

1.7 MRDSA System Relations

1.8 MRDSA System Procedures

1.9 MRDSA User and System Procedures used by "TABSINTEXT"

1.1 Introduction

"TABSINTEXT" is a program which implements the inclusion of tables into the body of a text. The text is preprocessed and information pertaining to each word is held in the form of a relation. Further to this, tags are placed in the text and are preprocessed in much the same way as non-tagged words. These tags serve as references to the location and placement of tables into the text. Data for each table are stored in relational form. These are set up before the program "TABSINTEXT" is executed.

The output from "TABSINTEXT" consists of an updated version of the table which holds the information about the words in the text, as well as a new relation having information pertaining to each table which has been incorporated into the corpus of the text.

Initially "TABSINTEXT" was written in MRDSA. MRDSA is a McGill Relational Database System implemented on the Apple II microcomputer using Apple Pascal. Though MRDSA is very powerful it taxes the APPLE to its limits. Compiling and running MRDSA programs is very slow since much swapping is required to accomodate the 75 K-byte system library into a 64 K-byte RAM.

"TABSINTEXT" was rewritten to use MRDS/FS which is a conversion and extension of MRDSA. MRDS/FS was developed by Ted Van Rossum for his master's Thesis and runs in the UCSD -p environment on the IBM PC with 128 K-bytes RAM and two double sided disk drives.

1.2 Introduction to MRDS/FS

MRDS/FS is a prototype of a tool for use in exploration of the concept of a relation as the primitive data unit. As such the system provides the user with a single data structure, the relation, plus a rich set of functions for the manipulation of that data structure. This tool can be used whenever individual pieces of data can be aggregated into one or more meaningful relations and the inter-relationships among these relations need be explored. Manipulation of the data as relations allows the user to interact with his data at a much higher level of abstraction than that provided by single value manipulations. This data format allows the data to deal with a set of inter-related values as a single unit, thereby greatly simplifying this type of complex programming task.

MRDS/FS is an interactive relational expression interpreter which provides the user with a complete set of relational programming functions such as the relational algebra functions, domain algebra functions, branching functions, and housekeeping functions. These functions allow the user to create complex user views of the database.

1.3 MRDSA

The McGill Relational Database System for the Apple microcomputer, (MRDSA), is a Pascal data sub-language built by George Chiu for a master's thesis in 1982. This system provides the user with a library of subroutines for the execution of relational and housekeeping functions such as project, mu-join, print, etc.

MRDSA manages its database by maintaining information on all relations in three system relations, REL, DOM, and RD. These relations are automatically created when a new database is set up. REL contains information on the name, size, and location in the database of each relation. DOM holds information on each domain, indicating name, size and virtual domain characteristics. RD ties REL and DOM together. That is, each domain in relation RD maintains information on the location in the tuple of the domain and the position of the domain in the sort order of the relation.

When the relation is constructed, (it may contain 1 to 50 diskettes), it is partitioned into 2 continuous sections. The first section contains all the relations which have been permanently saved, that is all system relations plus permanent relations are stored sequentially in contiguous sectors on the diskettes. The second section comprises any space remaining in the database and is used as workspace to store new relations created during an MRDSA session. These relations are discarded at the end of the session unless specifically saved.

MRDSA uses virtual memory system for accessing tuples of a relation. That is the database is partitioned into 512 K-byte pages, and required pages are read from disk into a group of cycling buffers using a FIFO demand paging system. This system allows MRDSA to handle arbitrarily large files.

Another important MRDSA feature is a full screen relational editor which allows the user to design his own screen layout, and update, input, or delete tuples in a relation. This editor is true to the relational concept in that it prevents the user from creating tuples with duplicate keys. This task is performed by first creating a C-directory of the values in the key, and then maintaining this directory during the edit session.

1.4 Running MRDSA on the IBM PC

Using MRDS/FS is tantamount to runnning an extended version of MRDSA on the IBM PC. All the MRDSA software is stored in a datafile called SYSTEM.LIBRARY in the boot diskette. In addition, the UCSD-p Pascal system unit PASCALIC is also stored in the above datafile. The following is the format of a MRDSA program...

```
Program format;
Uses globalerr,screenops,syspro,sort,insavedump,numer1,
     cnsdomop,fss_util,actual, krunch,algebra,project1,
     select1,forms,tidyfcn,process1,editor,rel_ops,
     recgnize,fss_hart,history;
                        :
                        :
   ** DECLARATION AND OTHER INTERNAL PROCEDURES DECLARATION **
                        :
                        :
 Begin (* format *)
     setup('','');
          :
          :
     ** PROGRAM BODY **
          :
          :
     promptboot;
 End; (* format *)
```

In the second line the declaration of the units (separate compilation modules) in MRDS/FS is made.

When we run the MRDSA program, the system will inquire for the following information :-

1. New database (Y/N):

2. Enter database name:

3. Enter number of diskettes (1..100):

4. Enter number of drives (1..6):

The first executable statement is always the call to the initialization procedure SETUP (see section 1.9). The first two items may be supplied as input parameters to SETUP, or if they are null (as shown in the example) the user is prompted.

If it is a new database, the system will create the required system relations, otherwise it will load the system relations from the database. The database name becomes the name of the diskette that contains the database. If the name of the database is EXAMPLE and there are three diskettes in all, then the diskettes bear the name EXAMPLE0, EXAMPLE1, and EXAMPLE2 respectively. The system works out for itself the size of the database. The first six pages are always reserved for storing the system relations recording the size of the database. All remaining pages after the last page of the database are for the workspace. The data would be stored in a datafile, MRDS.DATA . Finally, to utilize all available drives, MRDSA should know the number of disk drives which are available. The system will request the user to put the required diskette into an appropiate drive whenever it is necessary. These requests should be followed strictly in order to have the program running smoothly. However, this may mean the boot diskette may not be in the boot drive when the program finishes. Therefore, at the end of every MRDSA program PROMPTBOOT should be called, which will make sure the boot diskette is in the boot drive before the program terminates. Note that PROMPTBOOT could be omitted if the MRDSA procedure SAVE is used, which always calls PROMPTBOOT before terminating the program.

1.5 The Relational Approach

The relational approach to data is based on the realization that files that obey certain constraints may be considered as mathematical relations, and hence that elementary relational theory may be brought to bear on various practical problems of dealing with data in such files.

The following diagram shows some sample data in relational form.

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clarke	20	London
S5	Adams	30	New York

A table such as that in the diagram is referred to as a relation. Rows of such a table are generally referred to as tuples. Likewise, columns are usually referred to as attributes. A domain is a pool of values from which the actual value appearing in a given column are drawn.

"TABSINTEXT" views both the data for the text and the tables as a set of relations. This view provides a means for describing the data in its natural structure only without superimposing any additional structure for machine representation purposes. Each table can be simply represented as a relation, which in turn can be refined and processed to produce new relations. In addition, MRDSA provides a set of user and system procedures which allow the programmer to manipulate these relations.

In this relational data model, attribute relationships are represented by relations. Relations that represent associations can be existing relations in the database or they can be created using relational operators. These operators can be described using the relational algebra. The relational algebra is a collection of high-level operators on relations.

The two basic elements of the relational algebra as used by MRDSA are (1) Relations and (2) Attributes.

1. Relations

Relations are referred to by the name given to them by the programmer or by the name associated with them in the database. Relation names are STRING's in Pascal, eight bytes long. Relations are manipulated by MRDSA procedures when their names are specified as parameters. For example, an MRDSA procedure which creates a new relation must be supplied (as an input parameter) with the name of the new relation. All relations created by user procedures are temporary relations stored in the workspace.

2. Attributes

Attributes are referred to by the names given to them by the programmer or by the name associated with them in the database. Attribute names are STRING's in Pascal, eight bytes long and must be one of the existing domains. Attributes are used, in the context of the relation in which they occur, to control operations of the relational algebra. Attribute values are stored in character strings of length specified in the database or by the programmer.

1.6 MRDSA User Procedures

The following is a short description of each of the user-level procedures in MRDSA.

Name	Description
CONREL	Creates a constant relation on a given attribute.
DUMP	Dumps the three system control relations : REL, DOM, RD
EDIT	Invokes the relational editor to edit a relation.
MERJOIN	Creates the mu-join on two relations on specified attributes. MERJOIN includes the natural join and generalizes the set operations intersection, union and so forth.
PROJECT	Creates the projection of a relation on the specified attributes.
PRTREL	Outputs a relation in tabular form without reordering.

QTEXPR	Provides a query facility including quantifiers on single relations.
SAVE	Saves a set of relations on the user's permanent database and stops the run. A call to procedure SAVE, if made, should be the last statement in an MRDSA program (SAVE always calls PROMPTBOOT).
SETUP	Initializes MRDSA: must always be executed first in a program.
SIGJOIN	Creates a sigma join of two relations on specified attributes. SIGJOIN generalizes and replaces the divide and natural composition operations.

1.7 MRDSA System Relations

To understand most of the MRDSA system level procedures, it is necessary to know how MRDSA keeps house using the three system relations REL, DOM, and RD. These relations are always stored with each database and are loaded or created by SETUP. Each domain created adds to DOM and each relation created adds to REL and RD. They are searched by the routines FINDREL, FINDDOM, and FINDRD respectively. In order to include data about temporary relations, they are updated by special code in the system and user procedures but the updated version is not rewritten permanently to the master file. When the temporary relations is saved, the system relation on the master file are changed appropiately. All permanent and work relations are controlled by data in REL, DOM and RD.

REL (RNAME	WIDTH	SIZE	PAGE	RINDX	WINDX	INDXST	INVST)	index
REL	26	38	0	0	3	-1	-1	0
DOM	20	50	2	0	23	-1	-1	1
RD	8	126	4	0	19	-1	-1	2

RNAME is the name of each relation. WIDTH is the length of its tuple, in bytes. SIZE is the maximun permissible number of tuples. The maximum value of SIZE is 32767, the maximum value of a Pascal integer. PAGE is the address in virtual memory of the page containing the first tuple of the relation. RINDX is used by GETUPLE to record the next tuple to read in a sequential scan. Also WINDX gives the number of tuples in the relation. The index is used as a pointer by the internal procedures of MRDSA. RINDX < WINDX must be satisified before the next tuple is read and WINDX < SIZE before the next tuple is added.

DOM	(DNAME	LEN	OPER	LEFT	RIGHT	INDEX)	index
	RNAME	8		-1	-1	-1	0
	WIDTH	2		-1	-1	-1	1
	SIZE	2		-1	-1	-1	2
	PAGE	2		-1	-1	-1	3
	RINDX	2		-1	-1	-1	4
	WINDX	2		-1	-1	-1	5
	INDXST	2		-1	-1	-1	6
	INVST	2		-1	-1	-1	7
	DNAME	8		-1	-1	-1	8
	LEN	2		-1	-1	-1	9
	OPER	2		-1	-1	-1	10
	LEFT	2		-1	-1	-1	11
	RIGHT	2		-1	-1	-1	12
	INDEX	2		-1	-1	-1	13
	PRNAME	2		-1	-1	-1	14
	PDNAME	2		-1	-1	-1	15
	POS	2		-1	-1	-1	16
	SORTRANK	2		-1	-1	-1	17
	CINDX	2		-1	-1	-1	18
	XPOS	2		-1	-1	-1	19
	YPOS	2		-1	-1	-1	20
	LNLEN	2		-1	-1	-1	21

DNAME is the name of each domain. LEN is the length, in bytes, of the field representing the attribute. OPER, LEFT, RIGHT, and INDEX are not used by relational algebra operations, but will be used in the future for domain algebra operations. The additional domains CINDX, XPOS, YPOS, LNLEN are used in the Relational Editor.

RD (PRNAME	PDNAME	POS	SORTRANK)	index
0	0	1	-1	0
0	1	11	-1	1
0	2	13	-1	2
0	3	15	-1	3
0	4	17	-1	4
0	5	19	-1	5
0	6	21	-1	6
0	7	23	-1	7
0	8	25	-1	8
1	9	1	-1	9
1	10	11	-1	10
1	11	13	-1	11
1	12	15	-1	12
1	13	17	-1	13
1	14	19	-1	14
2	15	1	-1	15
2	16	3	-1	16
2	17	5	-1	17
2	18	7	-1	18

PRNAME is the index in REL of RNAME. PDNAME is the index in DOM of DNAME. POS is the position in the tuple of the first byte of the attribute. SORTRANK is the rank of the sort: if the value is -1 then the attributed is not sorted.

1.8 MRDSA System Procedures

The following is a brief description of the MRDSA system procedures.

Name	Description
ADMIN	Takes care of the setup phase of the user procedures PROJECT, SIGJOIN.
ADTUPLE	Sets integer pointers to the next available space for tuples in a relation.
CKADTUPLE	Used in MERJOIN and SIGJOIN to prevent the overwriting of useful tuples of the sorted relations by addition of new tuples resulting from the join.
CHECKIO	Writes out the I/O error number and stops the program when it occurs.
COMPARE	Compares two tuples for less, equal, great on given attributes.
ERROR	Writes out the error message and takes action according to the severity of the error.
FINDDOM	Given attribute list, finds indices in the system relation DOM.
FINDRD	Given relation name and, optionally, attribute list, finds indices in the system relation RD.
FINDREL	Given relation name, finds index in the system relation REL.
FORM	Designs the form template and takes care of the setup phase of the Relational Editor.
FREEZE	Puts size data into systen relation REL for a given relation.
GETPAGE	Retrieves a page of virtual memory, if necessary, to find a given tuple. Sets index

	of buffer in RAM where the page is loaded.
GETUPLE	Sets integer pointer to the next tuple to be read for a given relation.
LOCKTUPLE	Calls GETUPLE or ADTUPLE and then locks the buffer that holds the required tuple.
NEWDOM	Given domain list and domain lengths, add data to the system relation DOM.
NEWREL	Given name and attribute list for a new relation, adds data to system relations RD and REL. Must be followed by FREEZE.
OPENFILE	Opens the appropiate file (diskette) for I/O.
PAUSE	Halts the program temporarily and prompts the user to type space to continue.
PROCESSING	Implements all the tuple operations in the Relational Editor.
PSORT	Used in MERJOIN and SIGJOIN to sort the two operand relations.
PROMPTBOOT	Requests the user to put back the boot diskette in the boot drive.
READFILE	Reads a page in from the diskette.
RSORT	Uses external merge sort to sort a given relation on specified attributes.
SETDIRECTORY	Creates the C-directory in the Relational Editor.
SETS	Performs standard set operations on sets specified as integer arrays.
SETSORT	Sets up the SORTRANK in RD of the given relation.
STRG	Converts a given digit 0 - 9 to the corresponding character.
WRITEFILE	Writes a page to the diskette.

1.9 MRDSA User and System Procedures used by "TABSINTEXT"

The following is a detailed description of the MRDSA user and system procedures which are utilized by "TABSINTEXT".

1. PROCEDURE ADTUPLE

```
PROCEDURE ADTUPLE(RDPTR : INTEGER;VAR PTR,TPTR : INTEGER);
```

INPUT

RPTR -- Index in REL of the relation to which the new tuple is added.

OUTPUT

PTR -- Integer pointer points to the buffer containing the page to which the tuple is added.
TPTR -- Integer pointer points to the position of tuple in the buffer.

N.B. Usage

Call ADTUPLE to set PTR & TPTR, then fill in BUFPTR.

TECHNIQUE

Distinguish between constant or general relation, If general relation then call GETPAGE to set PTR.

DESCRIPTION

ADTUPLE allocates space for the next tuple, and returns two integer pointers, first to the index of the buffer where the page is loaded and second to the position of the tuple in the page. It is up to the programmer subsequently to fill in a character string, based on these pointers and of the right length, with the right information.

ADTUPLE may be of use to the programmer in performing tuple-by-tuple operations on relations directly. Note that when we want to access more than one tuple simultaneously, then LOCKTUPLE should be used. ADTUPLE increments attributes WINDX of the system relation REL.

2. PPOCEDURE EDIT

```
PROCEDURE EDIT(RNAME : STRING8;DOMLIST : DLIST;DOMLEN : INDEX;
               KEYNO,N : INTEGER;NEWNAME : STRING8;
               PROBEFACTOR,LOADFACTOR : REAL);
```

INPUT

RNAME -- name of input relation.
DOMLIST -- N attributes appear in order of search key attributes and the remaining attributes of relation RNAME.
DOMLEN -- N lengths corresponding to attributes of DOMLIST.
NEWRNAME -- Name of result relation.
PROBEFACTOR -- PROBEFACTOR >= 1, used in the construction of the C-directory.
LOADFACTOR -- LOADFACTOR <= 1, used in the construction of the C-directory.

NOTE

1. If RNAME is '' then a new relation NEWRNAME with the specification of DOMLIST and DOMLEN will be created.
2. Otherwise if RNAME <> NEWNAME then a new relation NEWRNAME will be created as an identical relation to relation RNAME.
3. If RNAME = NEWRNAME then no relation will be created but the relation RNAME will be changed by the set of tuples generated in the editing process.
4. In cases 2 & 3 specify only the search key attributes. In the DOMLIST, DOMLEN is ignored.

TECHNIQUE

1. Find the input relation and create new relation if necessary.
2. Find the form template or design one if relation has no associated template.
3. Respond to user command (Design or Process) until the user has finished.
4. Design : Design the form template.
5. Process : Create the C-directory and let the user edit the relation NEWRNAME.

DESCRIPTION

EDIT invokes the Relational Editor of MRDSA. The editor has two aspects, algebraic and interactive. Algebraically it is just another unary operator on relations like PROJECT or QT-expressions. However, the results is not determined algorithmically as in the other cases. The result depends on the interactive activity of the person editing the relation. Interactively the editor offers a number of features for creating or modifying a relation.

To achieve the best direct access performance, the programmer should always set the probe and load factors to 1. However if the number of partitions in the C-directory exceeds the implementation limit, then the programmer is advised to lower the load factor first. If this does not bring a reduction in the number of partitions, then the programmer should try to increase the probe factor. In any case the programmer should try to keep the factors as close to 1 as possible. In order to let the programmer have better control over the access performance, he can set either or both factors to negative values. Then the system will prompt for the factor and give the programmer the number of partitions and the average number of secondary access per direct access on the tuples. The programmer can try different sets of factors to obtain a satisfactory result. Then he can put the optimal factors into the parameters. Note that these things should be transparent to the end user and the programmer should try the above analysis again after the relation has been changed substantially.

3. FUNCTION FINDDOM

```
FUNCTION FINDDOM(DOMLIST : DLIST;N : INTEGER;
                 VAR DPTR : INDEX):BOOLEAN;
```

INPUT

DOMLIST - Array of N attribute names to be found.

OUTPUT

DPTR - Array of N indicies of the rows of DOM containing the names in DOMLIST.
Return true if not all attributes are found.

TECHNIQUE

FINDDOM returns the indices in the system relation DOM of a set of attributes.

4. FUNCTION FINDRD

```
FUNCTION FINDRD(RNAME : STRING8;DOMLIST : DLIST;
                VAR N : INTEGER;VAR RDPTR : INDEX) : BOOLEAN;
```

INPUT

RNAME - Name of relation to be found. (If N = 0, then all attribute names are to be found).
DOMLIST - Array of N attribute names to be found.

OUTPUT

RDPTR - Array of N indicies of RD.
Return true if error is found.

TECHNIQUE

If N = 0 set integer pointers to all rows of RD containing RNAME. Otherwise find the lower limit in RD of RNAME and do N sequential searches on unordered attributes, PRNAME, PDNAME of RD, from this limit to REL(2).Windx - 1.

5. FUNCTION FINDREL

FUNCTION FINDREL(RNAME : STRING8) : INTEGER;

INPUT

RNAME - Name of relation to be found.

OUTPUT

Return index of the rows of REL containing RNAME;
-1 if the relation is not found

TECHNIQUE

Sequential search on unordered attributes RNAME of REL.

DESCRIPTION

FINDREL returns the index in the system relation REL of a given relation.

6. PROCEDURE GETUPLE

```
PROCEDURE GETUPLE(RDPTR : INTEGER;CH : CHAR;
                  VAR PTR,TPTR : INTEGER);
```

INPUT

RPTR -- Index of relation in REL.
CH -- The change code, '' or 'c' : passed to GETUPLE.
'' means routine calling GETUPLE does not intend to change tuple.

OUTPUT

PTR -- Integer pointer points to the buffer containing the page to which the tuple is added.
TPTR -- Integer pointer points to the position of tuple in the buffer.

N.B. Usage

Call GETUPLE to set PTR & TPTR, then retrieve the tuple.

DESCRIPTION

GETUPLE locates the tuple in the requested page, and returns two integer pointers, the first to the index of the buffer where the page is loaded and the second to the position of the tuple in the page (that is, the first byte of the tuple). It is up to the user subsequently to read or to change a character string, based on these pointers and on the right length.

GETUPLE may be of use to the application programmer in performing tuple-by-tuple operations on relations directly. GETUPLE increments attribute RINDX of the system relation REL.

7. PROCEDURE PROMPTBOCT

PROCEDURE PROMPTBOOT;

DESCRIPTION

Since the boot disk drive may hold a database diskette at the end of a MRDSA run, PROMPTBOOT should be called to make sure the boot diskette is online. It should be always the last executable statement (unless SAVE is the last statement : SAVE calls PROMPTBOOT before terminating MRDSA).

8. PROCEDURE PRTREL

```
PROCEDURE PRTREL(RNAME : STRING8;TITLE : STRING80;
                 FILENAME : STRING14);
```

INPUT

RNAME -- Name of relation to be printed.
TITLE -- Printed at the top of page.
FILENAME -- Cne of the following
1. "CONSOLE:" : Output to Console
2. "PRINTER:" : Output to Printer
3. Text filename with format
"DISKETTENAME":"FILENAME" :
Output to the text file specified.

TECHNIQUE

1. Find RNAME in REL.
2. Find all attributes in RD.
3. If the destination is a text file then transfer the relation to the destinated file one tuple per line without any formatting; otherwise do the following.
4. Find POS, LEN of attributes and use their print control.
5. Provide at least 8 spaces per attribute; truncate if

necessary to one tuple per line.
6. Find attribute names as output to headers.
7. Output all tuples in order of appearance in the relation.
8. Output maximum 23 tuples per page on Console.

DESCRIPTION

PRTREL displays a relation on the console, or prints it to the printer or transfers it to a text file, depending on the value FILENAME. If FILENAME is null or is an illegal format then the procedure will prompt the user to enter the correct destination. It displays or prints the together with a title (if specified), the relation name (RNAME) and column headings (the attribute names). It displays or prints one tuple per line with one space between the columns (attributes) and truncates the tuples (and header lines) if they exceed 80 characters in width in their output layout. In case the output is to a text file; it will be transferred a tuple per line without formatting and the programmer will be prompted to put the right diskette in. Tuples are not reordered.

9. PROCEDURE SETUP

```
PROCEDURE SETUP(NEWDB : STRING1;DATABASE : STRING7);
```

INPUT

NEWDB -- "Y" for new database; "N" for old database.
DATABASE -- Name of database.

ACTION

1. Opens files, load REL, DOM, RD and process run parameters.
2. Request the necessary information : Database name, number of diskettes, and number of drives.

Note

This is always the first executable statement in a MRDSA program.

DESCRIPTION

This is MRDSA system initialization procedure. It is always the first executable statement in an MRDSA program. If one or both parameters are missing, the procedure will prompt the end user for the missing parameters. This option enables the programmer to specify parts of the required information through the parameters for the end user in using the editor.

```
 CCCCC   HH    HH   AAAAAAA   PPPPPPP   TTTTTTTT   EEEEEEEE   RRRRRRR
CC       HH    HH   AA    AA  PP    PP     TT      EE         RR    RR
CC       HHHHHHHH   AAAAAAA   PPPPPPP      TT      EEEEE      RRRRRRR
CC       HH    HH   AA    AA  PP           TT      EE         RR RR
 CCCCC   HH    HH   AA    AA  PP           TT      EEEEEEEE   RR    RR

                          22222
                       22222222222
                     2222        222
                                  222
                                2222
                             2222
                          2222
                       2222
                     222222222222222
                     222222222222222
```

2.1 Formatting Tables with Text.

2.2 The "INPUT" Relation, the "TABLES" Relation, and the Data to be referenced.

2.1 Formatting Tables with Text.

"TABSINTEXT" effectively incorporates tables into text in a manner which handles both the text, and the tables (which are to be incorporated into the text), as relations. When dealing with the placement and formatting of tables into text, it becomes necessary to adhere to a set of conventions for doing so. For instance, in the event that a table is too wide to fit on a page, then one cannot arbitrarily fold each row of the table in two. In addition, the material in each row (tuple) must be harmoniously aligned with other material in that column (domain) and so forth.

The page layout (the assignment of lines of text to pages while coping with figures, tables, footnotes etc.) is an important consideration and should consist of as many properly spaced lines on a page as will fit, while taking into account both the size and number of tables on the page. A list of some of these conventions is given in APPENDIX A.

2.2 The "INPUT" Relation, the "TABLES" Relation, and the Data for the Tables to be referenced.

The input to "TABSINTEXT" consist of two relations. The first, referred to as REL1 in the program and assigned the relation name "INPUT", holds the information pertaining to the text to be formatted.

The second relation, referred to as REL2 in the program, and assigned the relation name "TABLES", contains information relating to the one or more tables which are to be incorporated into the text. In fact, "TABLES" has no tuples at the start of processing, but a tuple is constructed and added to this relation whenever a reference to the table is made.

The data for each table to be encountered is also stored in relational form. As a result, this allows the program to access the MRDSA system relation REL so as to obtain information about the name, height and width of each table referenced. All permanent and work relations are controlled by the data in REL, DOM and RD, as was discussed in section 1.7.

The relations "INPUT", "TABLES", and those which refer to the tables which are to be referenced are derived using the Relational Editor.

The relation "INPUT" has the following format:

INPUT (Word, Seq, Wordleng, Line, Page)

The Significance of each attribute is as follows:

Word:- This attribute value gives the words listed in the text. Words may be tagged or not. A tagged word serves as an indicator to the placement of a table at that point in the text. Tagged words are easily differentiated from non-tagged words in that the first two characters in the word are "*T". In addition, the condition is set where a tagged word is the only word on any given line. The name of a table must be at most six (6) characters in length.

Seq:- This is the sequence of the words within aa given line. For tagged words, we do not care about the value of this attribute.

Wordleng:- This gives the length of the word, for text formatting purposes. For tagged words, we do not care about the value of this attribute.

Line:- This is the line number on which the word appears.

Page:- This is the page number on which the word appears. Initially, this attribute has no value. Later, "TABSINTEXT" outputs the page number attribute value, after it has determined the most appropriate page for the placement of the table and correspondingly, the most appropiate location of each line.

The following example gives a brief synopsis of the processing phase in which the relation "INPUT" is deduced from the body of the text. The text is as follows:

```
Line 1   The final array represents a relation which is
Line 2   said to be a projection of the following relation.
Line 3   Example : Consider the relation ORDER
Line 4   *T ORDER
Line 5   A permuted projection of this relation is as follows
Line 6   *T ORDER4
```

NB. *TORDER and *TORDER4 are references to tables to be placed into the text.

The relation "INPUT" would be as follows:

INPUT	Word	Seg	Wordlen	Line	Page
	The	1	3	1	0
	final	2	5	1	0
	array	3	5	1	0
	represents	4	10	1	0
	a	5	1	1	0
	relation	6	8	1	0
	which	7	5	1	0
	is	8	2	1	0
	said	1	4	2	0
	to	2	2	2	0
	be	3	2	2	0
	a	4	1	2	0
	projection	5	10	2	0
	of	6	2	2	0
	the	7	3	2	0
	following	8	9	2	0
	relation.	9	9	2	0
	Example	1	7	3	0
	:	2	1	3	0
	Consider	3	8	3	0
	the	4	3	3	0
	relation	5	8	3	0
	Supply	6	6	3	0
	*TSUPPLY	-	-	4	0
	A	1	1	5	0
	permuted	2	8	5	0
	projection	3	10	5	0
	of	4	2	5	0
	this	5	4	5	0
	relation	6	8	5	0
	is	7	2	5	0
	as	8	2	5	0
	follows	9	7	5	0
	*TSUPPROJ	-	-	6	0

The relation "TABLES" has the following format:

TABLES (Tname, Tsize, Twidth, Tpage, Tflag, Trank)

The significance of each attribute is as follows:

Tname:- This attribute holds the names of the tables which have been referenced in the text.

Tsize:- This gives the number of tuples (and hence tablesize) in the relation (table) corresponding to the Tname attribute. This is obtained from the size attribute of the MRDSA system relation REL.

Twidth:- This gives the length of "Tname" tuple (ie. the number of characters in the row of the table). This is obtained from the WIDTH attribute of REL.

Tpage:- This gives the number of the page on which the table was put.

Tflag:- The value of this attribute is either "T" or "B" corresponding to whether the table was put at the TOP or BOTTOM of the page.

Trank:- This gives the sequence number of the table on a page. There may be more than one table on a page.

The following example gives an indication of how the relation "TABLES" is derived:

Example:

Suppose the following tables were referenced during the processing of the text. Let us suppose that statistics for each table were collected, then "TABLES" would be as shown:

MARKS

STUDENT1	STUDENT2	ASS	EXAM
Brown	Brown	20.	50.
Hung	Hung	27.	58.
Jones	Jones	28.	62.
Raman	Raman	24.	66.
Smith	Smith	25.	60.

Tuple length = 30

Page on which the table was put = 2

Place of table on page = B

Seq. " " " " = 1

REGSTR

STUDENT1	COURSEMK
Smith	85.
Jones	90.
Browm	70.
Hung	85.
Raman	90.

Tuple length = 18

Page on which the table was put = 5

Place of table on page = T

Seq. " " " " = 1

FINAL

STUDENT1	COURSE
Brown	Aldat
Brown	Pascal
Hung	Algol68
Jones	Aldat
Jones	Algol68
Smith	APL
Smith	Pascal

Tuple length = 14

Page on which the table was put = 4

Place of table on page = 1

Seq. " " " " = 1

The resulting "TABLES" relation would look as follows:

TABLES

Tname	Tsize	Twidth	Tpage	Tflag	Trank
MARKS	5	30	2	B	1
REGSTR	5	18	5	B	1
FINAL	7	14	4	T	1

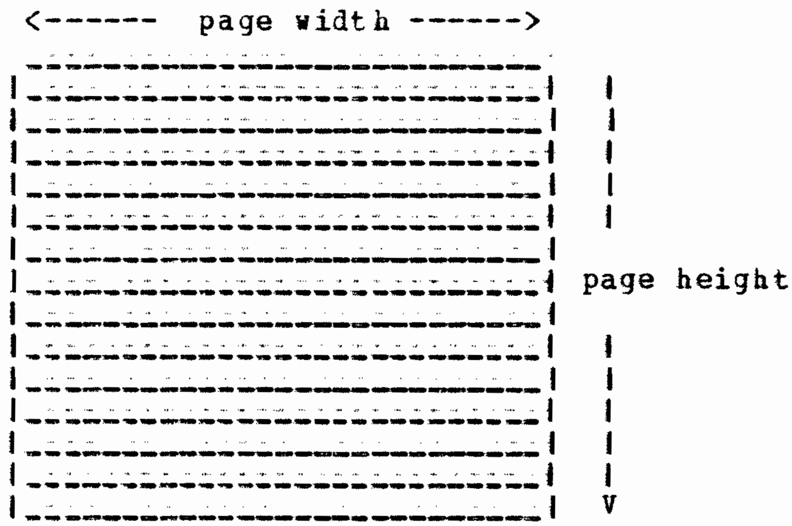
```
 CCCCC   HH    HH   AAAAAAA   PPPPPPP   TTTTTTTT   EEEEEEEE  RRRRRRR
CC       HH    HH   AA    AA  PP    PP     TT      EE        RR    RR
CC       HHHHHHHH   AAAAAAA   PPPPPPP      TT      EEEEE     RRRRRRRR
CC       HH    HH   AA    AA  PP           TT      EE        RR  RR
 CCCCC   HH    HH   AA    AA  PP           TT      EEEEEEEE  RR    RR
```

```
  33333333
333333333333
        333
        333
   33333333
   33333333
        333
        333
333333333333
  33333333
```

3.1 The Formatting Phase.

3.2 The Formatting Algorithm.

3.1 THE FORMATTING PHASE



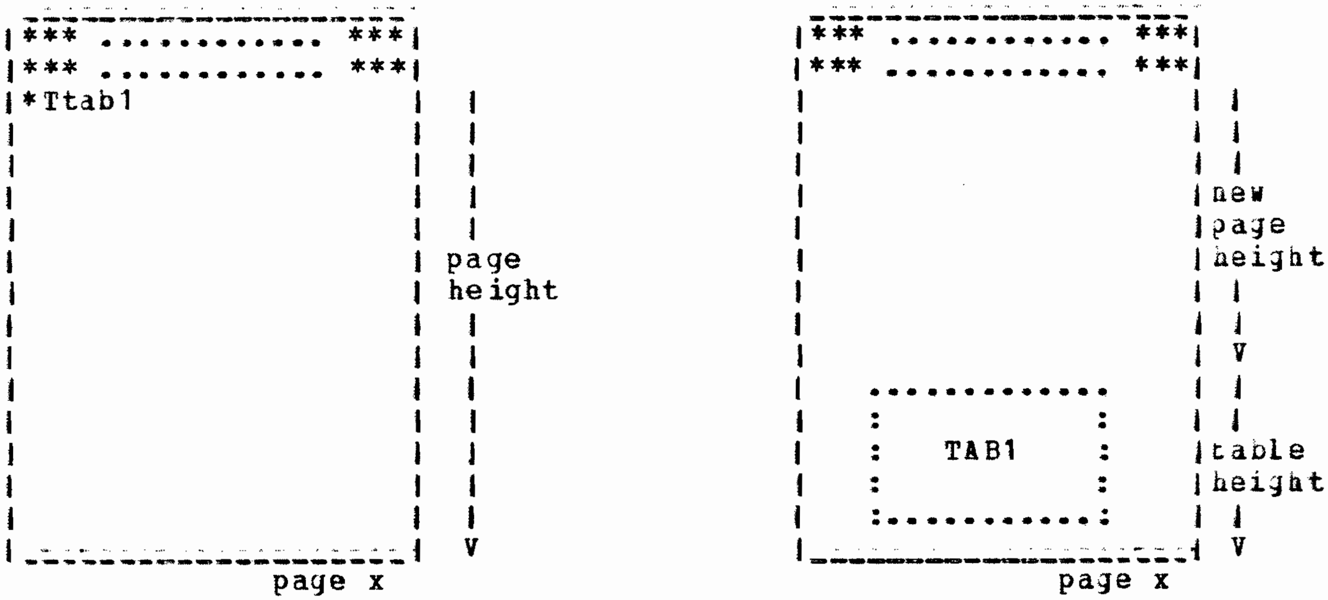
THE PAGE LAYOUT

As stated earlier, the page layout is an important consideration when including tables in text. The page frame, as used by "TABSINTEXT", has a page width which is a fixed unit, and a page height which is variable. Because "TABSINTEXT" does not allow pagebreaks, tables are not allowed either to exceed the width of the page or to cross over onto the next page from the current one.

The pageheight (which we define as the number of lines remaining on the page) varies between zero and a "stdheight" (a fixed height set up by the programmer to indicate the maximum number of lines to be allowed on the pages). Initially pageheight is equal to stdheight and as lines are placed onto the page, the pageheight is reduced until it becomes zero, at which time the page is full and no more lines can be put onto that page.

As soon as the reference to a table is encountered, the program attempts to output the table on the bottom of the page which is being formatted. This is possible if the height of the table (tableheight) is less than the pageheight, and is done by giving a value of "B" to the Tflag attribute in the tuple corresponding to the referenced table in the relation "TABLES". The following example illustrates this:

Suppose we are currently formatting page x, and the current pageheight = 20, height of table referenced = 12, then the table referenced (tab 1) is placed at the bottom of the page as follows:



The tuple corresponding to tab1 in the "TABLES" relation would be:

TABLES (Tname	Tsize	Twidth	Tpage	Tflag	Trank)
tab1	12	..	x	B	1

Suppose another table, tab2, was referenced on the same page as tab1, and that the height of tab2 was less than our new pageheight. Then tab2 will also be put on the "bottom" of the page, but it would appear above tab1 as follows:

```
 ---------------------------            ---------------------------
|***  ...........       ***|           |***  ...........       ***|
|***  ...........       ***|           |***  ...........       ***|
|*Ttab1                    |           |                          |  |
|***  ...........       ***|           |                          |new
|*Ttab2                    |           |                          |page
|                          |           |                          |height
|                          |           |                          |  |
|                          |           |                          |  V
|                          |           |    ............          |
|                          |           |    :   TAB2   :          |
|                          |           |    :..........:          |
|                          |           |    ............          |
|                          |           |    :          :          |
|                          |           |    :   TAB1   :          |
|                          |           |    :          :          |
|                          |           |    :..........:          |
|__________________________|           |__________________________|
                   page x                                 page x
```

The 'TABLES" relation would now be:

TABLES (Tname	Tsize	Twidth	Tpage	Tflag	Trank)
tab1			x	B	1
tab2			x	B	2

We can effectively interpret the Tflag/Trank attribute values as follows:

A Tflag/Trank value of B/1 implies that the respective table is the first table from the bottom of the page.

A Tflag/Trank value of B/2 implies that the respective table is the second table from the bottom of the page.

Now because the system does not allow pagebreaks (whenever the tableheight exceeds the pageheight), it may not always be possible to output the table at the bottom of the page currently being formatted. For instance, if a table happens to be twelve lines in length and there are only five lines on the current page, such a situation would arise. In this event, the table is output at the top of the next page. This is done by assigning a value of "T" to the Tflag attribute in the tuple corresponding to the referenced table in the "TABLES" relation. The following example illustrates this:

```
 ____________________                      ____________________
|***  ...........  ***|                   |***  ...........  ***|
|***  ...........  ***|                   |***  ...........  ***|
|*Ttab1               |                   |                    |  |
|                     |  |                |                    |  |
|                     |  |                |                    |  |
|                     |  |                |                    |new
|                     |  |                |                    |page
|                     | page              |                    |height
|                     | height            |                    |  |
|                     |  |                |   .............    |  V
|                     |  |                |   :           :    |  |
|                     |  |                |   :           :    |  |
|                     |  |                |   :   TAB1    :    |  |
|                     |  |                |   :           :    |table
|                     |  |                |   :           :    |height
|                     |  |                |   :...........:    |  |
|_____________________|  V                |____________________|  V
                 page x                                   page x

NB. There is enough
    room on page x
    for tab1
```

The TABLES relation would be:

```
 TABLES (Tname Tsize Twidth Tpage Tflag Trank)

         tab1  ..     ..      x     B     1
```

Later a reference is made to another table tab2 whose tableheight is less than the new page height.

```
 ---------------------              ---------------------
|***  ...........  ***|            |***  ...........  ***|
|***  ...........  ***|            |***  ...........  ***|
|*Ttab1               |            |                     |  |
|***  ...........  ***|            |                     |  |
|*Ttab2               |            |                     |  |
|                     |            |                     |current
|                     |            |                     |page
|                     |            |                     |height
|                     |            |                     |  |
|                     |            |   .............     |  V
|                     |            |   :           :     |  |
|                     |            |   :           :     |  |
|                     |            |   :   TAB1    :     |  |
|                     |            |   :           :     |table
|                     |            |   :           :     |height
|                     |            |   :...........:     |  |
|_____________________|            |_____________________|  V
               page x                             page x

                 ---------------------
                |    ..........       |
                |    :        :       |
                |    :        :       |
                |    :        :       |
                |    :        :       |
                |    :  TAB2  :       |
                |    :        :       |
                |    :        :       |
                |    :        :       |
                |    :        :       |
                |    :........:       |
                |                     | |
                |                     | |
                |                     |next page
                |                     |height
                |                     | |
                |_____________________| V
                               page x+1
```

The table relation would be:

TABLES (Tname	Tsize	Twidth	Tpage	Tflag	Trank)
tab1	...	....	x	B	1
tab2	...	....	x+1	T	1

Suppose a later reference is made to a third table tab3 whose tableheight is greater than the current pageheight but less than the next pageheight, then the following results:

```
 ------------------------           ------------------------
|***  ............  ***|           |***  ............  ***|
|***  ............  ***|           |***  ............  ***|
|*Ttab1                |           |                       |  |
|***  ............  ***|           |                       |  |
|*Ttab2                |           |                       |  |
|*Ttab3                |           |                       |current
|                      |           |                       |page
|                      |           |                       |height
|                      |           |                       |  |
|                      |           |    ..............     |  V
|                      |           |    :            :     |  |
|                      |           |    :            :     |  |
|                      |           |    :    TAB1    :     |  |
|                      |           |    :            :     |table
|                      |           |    :            :     |height
|                      |           |    :............:     |  |
|______________________|           |_______________________|  V
                 page x                             page x


                    ------------------------
                   |     ...........        |
                   |     :         :        |
                   |     :         :        |
                   |     :         :        |
                   |     :         :        |
                   |     :  TAB2   :        |
                   |     :         :        |
                   |     :         :        |
                   |     :         :        |
                   |     :.........:        |
                   |   ...............      |
                   |   :             :      |
                   |   :    TAB3     :      |
                   |   :             :      |
                   |   :.............:      |
                   |                        |  |next pageheight
                   |________________________|  V
                                   page x+1
```

The table relation would now be:

TABLES	(Tname	Tsize	Twidth	Tpage	Tflag	Trank)
	tab1	...	....	x	B	1
	tab2	...	....	x+1	T	1
	tab3	...	....	x+2	T	2

We can effectively interpret a Tflag/Trank attribute value of T/1 as the first table from the top of the page. A value of T/2 can be interpreted as the second from the top of the page and so forth.

This method of placing the table into pages may often result in a resequencing of the tables so that their order of appearance in the text differs from the order in which they were referenced. This is however traded off by the fact that "TABSINTEXT" seeks to place a table at the earliest possible position where it can fit.

To handle pagebreaks when they occur, "TABSINTEXT" sets up a linked list which is called the next-page-list. This list gives:

1) The next page number;

2) The next pageheight;

3) The number of tables already on that page.

To find the most appropiate page on which to put a table, the program searches down the next-page-list checking each record to see if the tableheight is less than the value of the pageheight field of that next page record.

When it encounters the first such a record, it updates the number of tables on the page and the pageheight items so as to reflect the placement of the table onto that page. In the event that no such record can be found, a new record is created, using the next page number after the last in the list, an updated pageheight (stdheight - tabheight) and an updated number of tables on page element. This new record is then added to the end of the next-page-list. After these operations, processing continues from the page which was being processed when the table was encountered.

Whenever there are no more available lines on the current page, the program accesses the next-page-list. If the list is not empty, then the values in the record at the top of the list are assigned to the current page values, and this top record is removed, hence effectively switching onto the next page, and still maintaining consistency with the upcoming "next" pages. If the list is empty however, the program adds one to the current page number and uses the stdheight value for the pageheight. An empty list implies that there are no tables, previously referenced, which are to be output on any upcoming page including and following the one presently being used.

3.2 THE FORMATTING ALGORITHM

The program "TABSINTEXT" is written in accordance with the algorithm given below. The boxes displayed throughout the algorithm outline the names of some of the routines through which the program proceeds.

```
| INITIALIZE |
|____________|
```

I a) Initialize Relations Attributes.

b) Initialize Next Page List.

c) Initialize Program Variables.

```
| PROCESS WORDS |
|_______________|
```

II a) Get the first tuple from the relation INPUT.

```
| PROCESS TUPLES |
|________________|
```

b) If the tuple is not a reference to a table then

```
| LOOP - ON - LINE |
|__________________|
```

1) While line number remains unchanged

a) Add page number to the page attribute of INPUT relation
b) If there are more tuples then get the next tuple.

2) Update line number value.

3) Go to II.d.

c) If the tuple is a reference to the table then

```
| PROC TABLES    |
|________________|
```

1) Determine the most appropriate page, and place on the page where the table can fit.

```
 ______________
| FINDAPPROPAGE |
|_______________|
```

2) Add a new tuple to the TABLES relation with the data which was derived from the referenced table.

```
 ______________
| UPDATE_TABLES |
|_______________|
```

3) Add page number on which the table was placed to the page attribute of the INPUT relation.

4) If there are more tuples, then get the next tuple.

d) Determine/Update the number of lines remaining on the current page.

e) If there are no more lines left on current page then

```
 ______________
| GET NEXT PAGE |
|_______________|
```

1) Get the next page values and replace the current page values with these.

III a) If there are more tuples in INPUT relation go to II.b.

```
 ___________
| PRINT RELS |
|____________|
```

b) Print relations INPUT, TABLES

c) STOP

The following illustration indicates how the algorithm proceeds:

A N E X A M P L E

Suppose we have the following INPUT relation

INPUT	Word	Seq	Wordlen	Line	Page
Tuple # 1	*Ttab1			1	0
2	Table1	1	6	2	0
3	is	2	2	2	0
4	an	3	2	2	0
5	example	4	7	2	0
6	of	5	2	2	0
7	the	6	3	2	0
8	INPUT	7	5	2	0
9	Relation	8	8	2	0
10	*Ttab2			3	0
11	Table2	1	6	4	0
12	is	2	2	4	0
13	an	3	2	4	0
14	example	4	7	4	0
15	of	5	2	4	0
16	the	6	3	4	0
17	TABLES	7	6	4	0
18	Relation	8	8	4	0

N.B. The field Tuple # has been indicated here purely for our convienence, and is not an attribute of the INPUT relation.

Suppose we have already gone through the Step I of the algorithm (the INITIALIZE phase) and that the following variable values have been obtained:

1. Pageno = 1
2. Lineno = 1
3. Currentpageht = 24
4. Number of tables on page (numtblsonpage) = 0

Let us make the following assumptions:

a. Tabheight = 20 for tab1
b. Tabheight = 15 for tab2.

Proceeding through the algorithm...

At Step II a : Get tuple 1

" " II c : Tuple 1 is a reference to tab1

" " II c.1 : Most appropiate page = 1 since currentpageht => tabheight for tab1

Position on page = B (bottom)

At Step II c.2 : Add new tuple to TABLES. The result is

TABLES

Tname	Tsize	Twidth	Tpage	Tflag	Trank
tab1	20	..	1	B	1

At Step II c.3 : Add pageno to page attribute of INPUT relation

INPUT

Word	Seq	Wordleng	Line	Page
*Ttab1			1	1
Table1	1	6	2	0
is	2	2	2	0

At Step II c.4 : There are more tuples ==> get the next tuple fr
the relation INPUT

At Step II d : No of lines on page ==>
currentpageht - tabheight = 24 - 20 = 4

:
:
:
:
:

At Step III : There are more tuples ==> go to II b

At Step II b : Tuple 2 is not a reference to a table

" " II b.1.a : Add pageno to page attribute in INPUT

INPUT

Word	Seq	Wordleng	Line	Page
*Ttab1			1	1
Table1	1	6	2	1
is	2	2	2	0

At Step II b.1.b : There are more tuples ==>

Get tuple 3

go to II b.1

```
    ______________________
   |                      |
   |                      V
After several iterations on line 2
                          |
   |______________________|
```

At tuple # 10 the INPUT relation looks like

INPUT

Word	Seq	Wordleng	Line	Page
*Ttab1			1	1
Table1	1	6	2	1
is	2	2	2	1
an	3	2	2	1
example	4	7	2	1
of	5	2	2	1
the	6	3	2	1
INPUT	7	5	2	1
Relation	8	8	2	1
*Ttab2			3	0

We are now at Step II b.

At Step II b.1 : Line no has changed (from 2 to 3)

" " II b.2 : Update lineno

" " II d : Number of lines on page ==>

currentpageht - 1 = 4 - 1 = 3

" " II e : There are more lines on page

At Step III a : Go to Step II b

At Step II c : Tuple 10 is a reference to a table

" " II c.1 : Most appropiate page = 2

(since tabheight > currentpageht)

" " II c.2 : Add new tuple to TABLES

TABLES

Tname	Tsize	Twidth	Tpage	Tflag	Trank
tab1	20	..	1	B	1
tab2	15	..	2	T	1

At Step II c.3 : Add pageno to page attribute in INPUT relation

INPUT

Word	Seq	Wordleng	Line	Page
*Ttab1			1	1
Table1	1	6	2	1
is	2	2	2	1
an	3	2	2	1
example	4	7	2	1
of	5	2	2	1
the	6	3	2	1
INPUT	7	5	2	1
Relation	8	8	2	1
*Ttab2			3	2

At Step II c.4 : There are more tuples in relation INPUT

==> Get the next tuple

" " II d : No update on the number of lines on current pag

" " II e : There are more lines on current page

At Step III a : Go to II b

At Step II b.1 : Tuple 11 is not a reference to a table

" " II b.1.a : Add page number to page attribute in INPUT

Tuple 11 becomes...

INPUT (tuple 11)

Word	Seq	Wordleng	Line	Page
Table2	1	6	2	1

At Step II b.1.b : There are more tuples

Get tuple 12; go to Step II b.1

```
    ____________________
   |                    |
   |                    V
After several iterations on line 4
                        |
   |____________________|
```

The INPUT relation is as follows:

INPUT

Word	Seq	Wordleng	Line	Page
*Ttab1			1	1
Table1	1	6	2	1
is	2	2	2	1
an	3	2	2	1
example	4	7	2	1
of	5	2	2	1
the	6	3	2	1
INPUT	7	5	2	1
Relation	8	8	2	1
*Ttab2			3	2
Table2	1	6	4	1
is	2	2	4	1
an	3	2	4	1
example	4	7	4	1
of	5	2	4	1
the	6	2	4	1
TABLES	7	6	4	1
Relation	8	8	4	1

and the TABLES relation is

TABLES

Tname	Tsize	Twidth	Tpage	Tflag	Trank
tab1	20	..	1	B	1
tab2	15	..	2	T	1

```
 CCCCC   HH    HH   AAAAAAA   PPPPPPP   TTTTTTTT   EEEEEEEE   RRRRRRR
CC       HH    HH   AA    AA  PP    PP     TT      EE         RR    RR
CC       HHHHHHHH   AAAAAAA   PPPPPPP      TT      EEEEE      RRRRRRR
CC       HH    HH   AA    AA  PP           TT      EE         RR  RR
 CCCCC   HH    HH   AA    AA  PP           TT      EEEEEEEE   RR    RR


                    4444    4444
                   4444     4444
                  4444      4444
                 4444       4444
                4444        4444
               4444         4444
              4444444444444444444444
              4444444444444444444444
                            4444
                            4444
```

4.1 The Program "TABSINTEXT"

4.2 Setting up the Input for "TABSINTEXT"

4.1 The Program "TABSINTEXT"

The program "TABSINTEXT" is a series of Pascal procedures, which alongside with its internal routines, calls upon several MRDSA procedures . The main program is embodied in the following lines of code ....

```
begin (* TABSINTEXT *)
     setup('N','TDATA');
     TABLEPROGRAM;
     promptboot;
end.   (* TABSINTEXT *)
```

The second line is a call to the MRDSA procedure SETUP and is the first executable statement in the program. The relations which are used by the program are stored in the database TDATA which is set up before the program is executed. Since the database already exist the first parameter in SETUP must be 'N'.

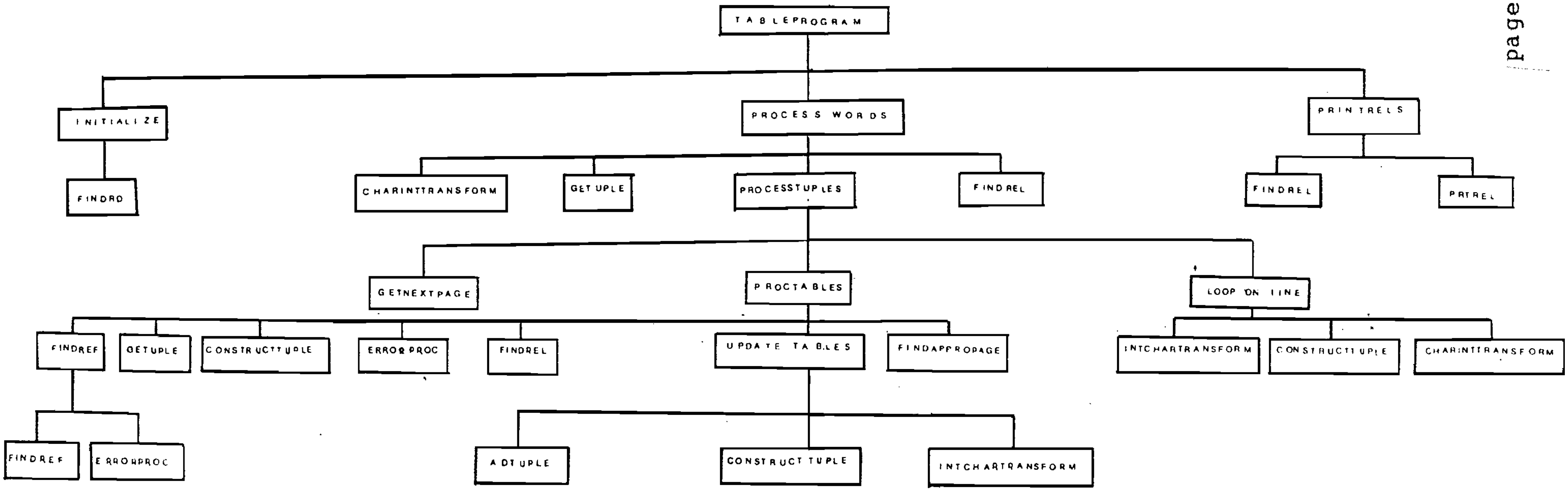
The third line is a call to the routine TABLEPROGRAM which controls all the processing done by TABSINTEXT. Specifically TABLEPROGRAM initializes the data (attribute names, sizes etc.), it calls the routine which processes the information in the relation INPUT, and finally it prints the INPUT and TABLES relations.

The MRDSA procedure PROMPTBOOT is called at line number four and it request the user to put back the boot diskette in the boot drive.

The procedure TABLEPROGRAM and indeed the overall flow of the program is shown by the flowchart below.

```
 ...........
 :  START  :
 :.........:
      |
      |
      V
 .............
 : INITIALIZE :
 :............:
      |
      |
      V
 ...............
 : PROCESSWORDS:
 :.............:
      |
      |
      V
 ............
 : PRINTRELS:
 :..........:
      |
      |
      V
 .........
 : RETURN:
 :.......:
```

The hierarchy of procedure calls within TABLEPROGRAM is shown in the diagram which follows.



PROCEDURE CALLS
WITHIN
TABLEPROGRAM

INITIALIZE

This internal procedure specifies the names of the attributes in the relaticns INPUT and TABLES and then locates (using the MRDSA function FINDRD and the system relation RD) the starting positions of each of the attributes in the tuples of the corresponding relations. From this is deduced the length of each attribute. These values are useful when it comes to adding specific atributes into these tuples since the tuple itself is but a string of characters.

PRINTRELS

PRINTRELS generates the final output from the program. It uses the MRDSA function FINDREL and procedure PRTREL to locate and print out the updated versions of both the INPUT and TABLES relations. Before printing occurs the RINDX field of the system relation REL has to be set to zero since during the course of processing this field would have been used by GETUPLE to record the next tuple to be read in sequential manner. Setting RINDX to zero would in effect set the pointer to the first tuple so that reading and hence printing could commence from the top of the relation.

PROCESSWORDS

As the name would indicate this routine processes the words in the text. Since the text is held in relational format, these operations are implemented by a call to the procedure PROCESSTUPLES which in turn operates on each tuple in the relation INPUT. PROCESSWORDS begins by finding the first tuple in this relation and thereupon derives the starting line number value. To obtain these, calls are made to the MRDSA function FINDREL and to the procedure GETUPLE as well as to the internal routine CHARINTTRANSFORM which transforms a character string into its equivalent numeric value.

INTCHARTRANSFORM

Because all elements of the relations in the database are stored as characters it is necessary to transform integers into their equivalent character format. This procedure accomplishes this task for integer of eight digits or less.

CHARINTRANSFORM

CHARINTTRANSFORM accomplishes the opposite of INTCHARTRANSFORM in that it converts a string of characters between 0 and 9 into its numeric equivalent. This is necessary so as to retrieve numeric value data from the relation INPUT.

CONSTRUCTTUPLE

This routine is used to construct a tuple by filling in a single attribute each time it is called. The parameters for this rcutine are as follows .....

p1 = Integer pointer to buffer containing the page.

p2 = " " " position of tuple in page.

x = length of attribute in tuple.

y = position of attribute in tuple.

str= string holding attribute value.

The routine proceeds by filling each character of the string into the text field of the record for the work file of the particular relation.

PROCESSTUPLES

Before the processing of the tuples of the INPUT relation begins, PROCESSTUPLES initializes the program variables which are to be used in this phase. These include the following ....

tabnum -------- table number; initially set to 0

pageno -------- page number; initially set to 1

numtuples ----- number of tuples; set at 1
since tuple 1 has already been obtained

currentpageht -- current page height; set to stdheight

numtblsonpg ---- number of tables on page; initially set to 0

inputsize ------ number of tuples in INPUT

For each line of text PROCESSTUPLES operates on all tuples with data pertaining to that specific line. When a new line is encountered the first word of that line is checked for a table reference. This is done by checking the first two positicns of the WORD attribute for the characters '*' and 'T'. If a reference to a table is indicated the procedure PROCTABLES is called else a loop is entered in which all tuples pertaining to that line are processed until a new line is found. For each line that is processed the current page height is appropiately adjusted. When this height becomes zero a call is made to the procedure GETNEXTPAGE which finds the next page unto which processing can resume.

GETNEXTPAGE

when all the lines on the current page have been used, this routine is called to find the next page as well as to obtain the relevant information pertaining to that page. In order to achieve this objective, GETNEXTPAGE attempts to access the next-page list (discussed in section 3.1). If the list is empty then the page number is increased by 1, the current page height is set to stdheight, and there would be no tables on the upcoming page. If the list is not empty however, the top element is removed and the page number and other relevant information are obtained from fields in this record. The subsequent record now becomes the top record of the list.

LOOP_ON_LINE

If the first word on a line does not refer to a table then this procedure is invoked. It is this routine that processes the tuples which pertain to an individual line of text. This includes the addition of the page number to the PAGE attribute of the relation INPUT, and if the tuple number is less that the total number of tuples in the relation then the next tuple is obtained and the line number is deduced from that tuple. In order to accomplish these tasks LOOP_ON_LINE calls INTCHARTRANSFORM, CHARINTRANSFORM, CONSTRUCTTUPLE and the MRDSA routine GETUPLE.

PROCTABLES

This procedure is called whenever a reference to a table is made. It begins by finding the indices in the system relation REL of the relations TABLES and of the tables to be referenced. The latter is accomplished by the procedure FINDREF. PROCTABLES then checks the height and width of the referenced table to determine if it can fit on a page. If it cannot an error message is printed by the procedure ERRORPROC and the program is halted. If the table can fit and can do so on the current page being processed then the number of tables on the page is increased by one, the table is flagged to be put at the bottom of the page, the sequence of the table on the page is found, the current page height is adjusted to reflect the placement of the table onto that page, and the relations INPUT and TABLES are appropiately adjusted by calls to UPDATE_TABLES in the case of the TABLES relation and CONSTRUCTTUPLES in the case of the INPUT relation. If the table is not able to fit on the current page then it is placed at the top of the next page if there are no tables on the upcoming "next" pages, or is placed in sequence (by rank) on the most appropiate page. Finally PROCTABLES checks if there are more tuples in INPUT and if so then it gets the next tuple so that further processing can take place.

FINDREF

FINDREF locates the index in the system relation REL of the tables which are referenced. The routine makes use of the MRDSA function FINDREL to accomplish this. The table names must be specified by the user as the parameter to FINDREL and the tables must be listed within the 'case' statement in the exact order in which they appear in the text. This is because the "tabnum" variable keeps track of the number of tables referenced and in order to find the correct index in REL, "tabnum" must match the case label. If the user does not specify all the tables within the routine then a call is made to ERRORPROC and a message is signalled that a table has been excluded from the list before the program halts processing.

UPDATE_TABLES

Having derived all the necessary information for a particular table that has been referenced, a tuple is now compiled (attribute by attribute) and added to the relation TABLES. Calls are made to the MRDSA procedure ADTUPLE and to CONSTRUCTTUPLE, INTCHARTRANSFORN, and CHARINTTRANSFORM.

FINDAPPROPAGE

In the event that a table cannot fit on a page this routine is called to find the most appropiate page onto which that table can be placed. This is done by searching down the next-page list until a record is found where the pageheight field is greater than the height of the table. At this point the table is effectively placed on that page by updating the pageheight, tblsonpg (tables on page), pnum (page number) and rank fields of that record. If the end of the list is reached and no such record has been found then a new record is created and the necessary information are placed into the fields of that record. This is then added to the bottom of the next-page list.

ERRORPROC

This routine is called whenever the program recognizes an error in the data. Three errors have been specified and include

1. A table being toc long to fit on a page,
2. A table being excluded from the list in the procedure FINDREF, and
3. A table being too wide to fit on a page.

After printing the appropiate message, execution is halted so that the user can take whatever corrective measures that are necessary.

4.2 Setting up the Data for "TABSINTEXT"

The data for the relations which are input to TABSINTEXT are set up using the Relational Editor. In order to use the Editor a MRDSA program must be written to call the user procedure EDIT, which in turn invokes the Editor. The Editcr was designed tc provide a high level interface for the end user.

The program SETUP_INPUT is typical of how the input should be set up. The variables "domlist1" and "domlist2" refer to the INPUT and TABLES relations respectively, whereas "tab1" to "tabn" refer to the first to the n-th table to be referenced.

The procedure EDIT is invoked so that the relations could be created and edited. PRTREL is subsequently called to print out the relations whilst SAVE permanently saves them on the database.

Appendix C gives a listing of the program SETUP_INPUT and Appendix D gives a sample of the data used by the relation INPUT and TABLES as well as for the tables to be referenced. It should be noted, in the case of the TABLES relation, that ideally this relation should initially have no tuples. However MRDSA does not appear to recognize and save null relations. This problem is resolved by having the first tuple of TABLES be a series of zeroes or "don't care" characters and this tuple is disregarded in subsequent dealings with this relation.

The INPUT and TABLES relations which are output from the program are given in Appendix E.

```
AAAAAA   PPPPP    PPPPP    EEEEEE  N      N  DDDD    IIIIIIIII X       X
A    A   P    P   P    P   E       NN     N  D   D       I       X     X
A    A   P    P   P    P   E       N N    N  D    D      I        X X
AAAAAA   PPPPP    PPPPP    EEEE    N  N   N  D    D      I         X
A    A   P        P        E       N    N N  D    D      I        X X
A    A   P        P        E       N     NN  D   D       I       X     X
A    A   P        P        EEEEEE  N      N  DDDD    IIIIIIIII X       X
```

```
 AAAAAAAA
AA      AA
AA      AA
AAAAAAAAAA
AA      AA
AA      AA
AA      AA
```

The following is a list of standard coventions for coping with figures, tables, footnotes, etc in text.

1) The purpose of tabulation is to present data more vividly and concisely than is possible in the text. Tables can often be condensed:- a factor common to all elements in a column can be incorporated into a column heading; a variable pertaining to only one or a few entries in a large series can be indicated in a footnote (Table 2).

2) Table titles should be brief. Explanation, if needed, should be given in a footnote.

3) When tables are referred to in the text, they should be numbered consecutively throughout the work, not beginning a new series of numbers with each new chapter. Reference in the text should be to table number, not to a specific page. The table number may be either Roman or Arabic numerals and may either be set on a separate line or run in with the caption.

Example: TABLE II

Marsh Herbs

Table 2: Immigrant Aliens Admitted to the United States

4) Table numbers and captions are usually set above the table itself.

5) Short tables are clearer and more forceful than long ones. A large, unwieldly table, therefore, should be broken up into separate smaller tables if the data will allow.

6) Use zero to indicate "none" in answer to the implied question "how much?" or "how many?" (Table 2).

7) Use ellipses to indicate that no data were available or that a specified category of data is not applicable.

8) If all entries in any one column are expressed in decimal fractions less than one, zeroes must be used before the cipher. Decimal points should be aligned.

9) In Stubs (first entry in horizontal columns) and their subdivisions, capitalize only the first word (Table 2).

TABLE 2 -- COMMON MANIFESTATIONS ON COLRECTAL CANCER

Manifestations	Cattel	Hallstand	Palumbo	Gilchrist
Changes in bowel habits	69.3	...	72.75	50.0
Pain	68.0	69.96	81.25	24.0
Anenia	20.6	4.35	0.0	12.0
Weight Loss	50.6	76.28	66.25	6.0 *
Obstruction	0	10.27	25.00	10.0
Asymptomatic	0	0	...	7.0

APPENDIX A

```
AAAAAA   PPPPP    PPPPP    EEEEEE  N      N  DDDD    IIIIIIIII X      X
A    A   P    P   P    P   E       NN     N  D   D       I       X   X
A    A   P    P   P    P   E       N N    N  D    D      I        X X
AAAAAA   PPPPP    PPPPP    EEEE    N  N   N  D    D      I         X
A    A   P        P        E       N   N  N  D    D      I        X X
A    A   P        P        E       N    NN   D   D       I       X   X
A    A   P        P        EEEEEE  N      N  DDDD    IIIIIIIII X      X

                              BBBBBBB
                              BB    BB
                              BB    BB
                              BBBBBBB
                              BB    BB
                              BB    BB
                              BBBBBBB
```

The following is a listing of the program "TABSINTEXT".

```
(*$s++,n++*)
Program tabsintext(input,output);

  uses globalerr, screenops, syspro, sort, insavedump, algebra;

  const stdheight = 24;      { max. number of lines per page }
        stdwidth  = 70;      { max. number of characters per line }

  type  domstr =  packed array[1..8] of char;
        tablinks=^tabref;
           tabref=record
                  pagenum,pageheight,tblsonpg : integer;
                  nextpage : tablinks
                  end;
        strng8 = array [1..8] of char;

   var  pword,pseq,pwordlength,pline,
        ppage,ptname,ptsize,ptwidth,ptpage,ptflag,ptrank,
        lword,lseq,lwordlength,lline,lpage,ltname,ltsize,ltwidth,
        ltpage,ltflag,ltrank,tabnum,numtuples,inputsize,
        rptr1,ptr1,tptr1       :    integer;
        ch:char;


Procedure intchartransform(num,numdigits :integer; var s:domstr);
        var indx,rem : integer;

     (* THIS ROUTINE TRANSFORMS AN INTEGER INTO A CHARACTER STRING.
        THIS IS BECAUSE ALL ELEMENTS OF THE RELATION ARE STORED
        IN CHARACTER FORMAT.   *)

     begin (* intchartransform *)
         for indx := 1 to 8 do s[indx] := ' ';
         for indx := numdigits downto 1 do
            begin
               rem := num mod 10;
               case rem of
         0:       s[indx]:='0';
         1:       s[indx]:='1';
         2:       s[indx]:='2';
         3:       s[indx]:='3';
         4:       s[indx]:='4';
         5:       s[indx]:='5';
         6:       s[indx]:='6';
         7:       s[indx]:='7';
         8:       s[indx]:='8';
         9:       s[indx]:='9';
               end;
               num := num div 10;
            end;
     end;  (* intchartransform *)
```

APPENDIX B

```
Procedure charinttransform(var sum:integer;  s:domstr);

  var exp,indx1,k,v : integer;

   (*  THIS ROUTINE CONVERTS A STRING OF CHARACTERS BETWEEN
       0 AND 9 INTO ITS NUMERIC EQUIVALENT *)

  begin (* charinttransform *)
      sum:=0;
      for indx1:= 8 downto 1 do
         begin
         exp:=1;
         k:= 8 - indx1;
         if k>0 then
            begin
                for v:= 1 to k do exp:=exp*10;
            end;
         case s[indx1] of
    '1':       sum:=sum + (1*exp);
    '2':       sum:=sum + (2*exp);
    '3':       sum:=sum + (3*exp);
    '4':       sum:=sum + (4*exp);
    '5':       sum:=sum + (5*exp);
    '6':       sum:=sum + (6*exp);
    '7':       sum:=sum + (7*exp);
    '8':       sum:=sum + (8*exp);
    '9':       sum:=sum + (9*exp);
         end;
     end;
 end; (* charinttransform *)
```

```
Procedure constructtuple(var p1,p2,x,y:integer; var str :domstr);

   var indx,indx1 : integer;

    (*  THIS ROUTINE IS USED TO CONSTRUCT A TUPLE BY FILLING
        IN A SINGLE ATTRIBUTE EACH TIME IT IS CALLED. THE
        PARAMETERS ARE AS FOLLOWS ....
            p1 = INTEGER POINTER TO BUFFER CONTAINING THE PAGE
            p2 =    "           "       " POSITION OF TUPLE IN THE PAGE
            x  = LENGTH OF ATTRIBUTE IN THE TUPLE
            y  = POSITION OF ATTRIBUTE IN TH TUPLE
            str = STRING HOLDING ATTRIBUTE VALUE                   *)

    begin (* constructtuple *)
    writeln('enter  constructtuple');
        indx := p2;
        for indx1 := 1 to x do
            begin
                bufptr[p1]^.data[indx+y-1] := str[indx1];
                indx := indx+1;
            end;
    end;  (* constructtuple *)


 Procedure errorproc(err :integer;tabname :domstr);
     begin (* errorproc *)
        case err of
        1: begin
              writeln;writeln;writeln;
              writeln(tabname,' is to long to fit on page.');
              writeln;writeln;
              halt;
           end;
        2: begin
              writeln;writeln;writeln;
              writeln(tabname,' is excluded from list in proc FINDREF');
              writeln;writeln;
              halt;
           end;
        3: begin
              writeln;writeln;writeln;
              writeln(tabname,' is to wide to fit on page.');
              writeln;writeln;
              halt;
           end;
        end
      end;  (* errorproc *)
```

APPENDIX B

```
Procedure proctables(tabname:domstr; toppage:tablinks;
                        var numtblsonpg,currentpageht,pageno:integer);

   var k,tabheight,tabptr,tabwidth,refptr,err,pnum,rank : integer;
       c,flag : char;
       tabstr : strng8;
       postr : domstr;
       st : packed array[1..8] of char;

 Procedure findappropage(var pnum :integer);
   var b : boolean;
       pg,npg : tablinks;

    begin   (* findappropage *)
    writeln('enter findappropage');
       new(pg);
       pg:=toppage;
       b:=true;
       while (pg <> nil) and (b = true)  do
           begin
               if tabheight<=pg^.pageheight then
                       begin
                           pg^.pageheight:=pg^.pageheight - tabheight;
                           pg^.tblsonpg:=pg^.tblsonpg + 1;
                           pnum:= pg^.pagenum;
                           rank:= pg^.tblsonpg;
                           b:= false;
                       end
               else
                   if pg^.nextpage=nil then
                       begin
                           b:=false;
                           new(npg);
                           npg^.pagenum:=pg^.pagenum+1;
                           npg^.pageheight:=stdheight-tabheight;
                           npg^.tblsonpg:=1;
                           npg^.nextpage:=nil;
                           rank:=1;
                           pnum:=npg^.pagenum;
                           pg^.nextpage:=npg;
                       end
                   else
                       pg:=pg^.nextpage;
           end;
           if (pg = nil) and (b = true) then pnum := 0;
      end; (* findappropage *)
```

```
Procedure update_tables(var pgno:integer);

    var indx,ptr2,tptr2 : integer;
        nstr : domstr;


    begin (* update_tables *)
    writeln( enter update_tables');
        adtuple(tabptr,ptr2,tptr2);
        constructtuple(ptr2,tptr2,ltname,ptname,tabname);
        intchartransforn(tabheight,ltsize,nstr);
        constructtuple(ptr2,tptr2,ltsize,ptsize,nstr);
        intchartransform(tabwidth,ltwidth,nstr);
        constructtuple(ptr2,tptr2,ltwidth,ptwidth,nstr);
        intchartransform(pgno,ltpage,nstr);
        constructtuple(ptr2,tptr2,ltpage,ptpage,nstr);
        intchartransform(rank,ltrank,nstr);
        constructtuple(ptr2,tptr2,ltrank,ptrank,nstr);
        nstr[1]:=flag;
        for indx := 2 to 8 do nstr[indx] := ' ';
        constructtuple(ptr2,tptr2,ltflag,ptflag,nstr);
    end;   (* update_tables *)



    Procedure findref;

       begin
       writeln('enter findref');
           refptr:=0;
           case tabnum of
       1   :  refptr:=findrel('ORDER');
       2   :  refptr:=findrel('ORDER4');
       3   :  refptr:=findrel('MARK');
       4   :  refptr:=findrel('FINAL');
       5   :  refptr:=findrel('REGSTR');
       6   :  refptr:=findrel('TBLEX');
           end;
           if refptr=0 then
              begin
                 err:=2;
                 errorproc(err,tabname);
              end;
       end; (* findref *)
```

APPENDIX B

```
begin (* proctables *)
     tabptr:=findrel('TABLES');
     tabnum:=tabnum+1;
     findref;
     tabheight:=rel[refptr].winox;
     if tabheight>stdheight then
        begin
           err:=1;
           errorproc(err,tabname);
        end;
     tabwidth:=rel[refptr].width;
     if tabwidth>stdwidth then
        begin
           err:=3;
           errorproc(err,tabname);
        end;
     if tabheight<=currentpageht then
        begin
           flag:='b';
           numtblsonpg:=numtblsonpg+1;
           currentpageht := currentpageht-tabheight;
           rank := numtblsonpg;
           update_tables(pageno);
           intchartransform(pageno,lpage,pgstr);
           for k := 1 to 8 do st[k]:=pgstr[k];
           constructtuple(ptr1,tptr1,lpage,ppage,st);
        end
     else
        begin
           if toppage = nil then
              with toppage^ do
                 begin
                    pagenum:=pageno + 1;
                    pageheight:= stdheight;
                    rank:=0;
                    nextpage:=nil;
                 end;
           flag:='t';
           findappropage(pnum);
           if pnum = 0 then pnum := pageno + 1;
           update_tables(pnum);
           intchartransform(pnum,lpage,pgstr);
           for k:= 1 to 8 do st[k]:=pgstr[k];
           constructtuple(ptr1,tptr1,lpage,ppage,st);
        end;
    if numtuples < inputsize then
        begin
           c:= c ;
           getuple(rptr1,c,ptr1,tptr1);
        end;
    numtuples:=numtuples+1;
end;  (* proctables *)
```

APPENDIX B

```
Procedure processwords;
   var lineno,pageno,indxi,dummy : integer;
       lnstr,postr : domstr;
       c : char;
       st : packed array[1..8] of char;



   Procedure processtuples;

      var linecount,currentpageht,numtblsonpg,
          k,indx : integer;
          wordstr : strng8;
          toppage : tablinks;
          tabname : domstr;
```

APPENDIX B

```
Procedure loop_on_line;

   (*  IF THE FIRST WORD ON A LINE DOES NOT REFER TO A TABLE THEN
       THIS PROCEDURE IS INVOKED. THIS ROUTINE PROCESSES THE TUPLES
       WHICH PERTAIN TO WORDS IN AN INDIVIDUAL LINE. PROCESSING IN
       THIS ROUTINE ENDS WHEN THE FIRST WORD OF THE NEXT LINE IS
       ENCOUNTERED. THAT IS, WHEN THE LINE NUMBER CHANGES. *)

     var npstr : strng8;

  begin
      intchartransform(pageno,lpage,pgstr);
      for indx1:= 1 to 8 do st[indx1]:=pgstr[indx];
      while (lineno = linecount) and (numtuples<=inputsize) do
      begin

         (* add pageno to the PAGE attribute of the
            relation INPUT  *)


         constructtuple(ptr1,tptr1,lpage,ppage,st);

         (* get the next tuple and obtain the value of the
            line number in the LINE attribute *)

         c := 'c';
         if numtuples < inputsize then getuple(rptr1,c,ptr1,tptr1);
         numtuples:=numtuples+1;
         dummy:=tptr1;
         for indx1 := 1 to 8 do lnstr[indx1]:=' ';
         for indx1 := (9 - lline) to 8 do
             begin
                 lnstr[indx1] := bufptr[ptr1]^.data[dummy+pline-1];
                 dummy:=dummy+1;
             end;
         charinttransform(lineno,lnstr);
         writeln;writeln('lineno = ',lineno);writeln;
     end;
  end; (* loop_on_line *)
```

APPENDIX B

```
Procedure getnextpage;
  (*  WHEN ALL THE LINES ON THE CURRENT PAGE HAVE BEEN USED  THIS
      ROUTINE IS CALLED TO FIND THE NEXT PAGE AS WELL AS TO OBTAIN
      THE RELEVANT INFORMATION PERTAINING TO THAT PAGE.   *)


    var b : boolean;
        pg : tablinks;

     begin (* getnextpage *)
         new(pg);
         pg:=toppage;
         b:=true;
         if pg = nil then
            begin
                currentpageht := stdheight;
                pageno := pageno + 1;
                numtblsonpg :=0;
            end
         else
            begin
                pageno := pg^.pagenum;
                currentpageht := pg^.pageheight;
                numtblsonpg := pg^.tblsonpg;
                toppage := pg^.nextpage;
                dispose(pg);
             end;
     end; (* getnextpage *)
```

APPENDIX B

```
begin (* processtuples *)
    tabnum:=0;
    numtuples := 1;
    currentpageht := stdheight;
    numtblsonpg := 0;
    toppage := nil;
    linecount := lineno;
    inputsize := rel[rptr1].windx;
    while numtuples<inputsize do
        begin

            (* process all tuples with data pertaining to a
               specific line. For a new line, reset linecount and
               decrease current pageheight  *)

            linecount := lineno;

            (* check the first word of this new line for a table
               reference  *)

            dummy := tptr1;

            (* obtain the character string of the first word of the
               new line  *)

            for indx1:= 1 to 8 do
               begin
                   writeln(indx1);
                   wordstr[indx1]:=' ';
               end;
            for indx1 := 1 to 8 do
               begin
                   wordstr[indx1] := bufptr[ptr1]^.data[dummy+pword-1];
                   dummy:=dummy + 1;
               end;

            (* check for table reference *)

            writeln;writeln;
            write('                ');
            for indx1:=1 to 8 do write(wordstr[indx1]);
            writeln;writeln;read(ch);
```

```
    if (wordstr[1]='*') and (wordstr[2]='[') then
       begin
          for k:= 1 to 8 do tabname[k]:=' ';
          for k := 3 to 8 do
             begin
                indx:=k-2;
                tabname[indx]:=wordstr[k];
             end;
             writeln;
             writeln('            tabname is ',tabname:8);
             writeln;
         proctables(tabname,toppage,numtblsonpg,currentpageht,pager
o);

        end
    else
        begin
           loop_on_line;
           currentpageht := currentpageht - 1;
        end;


    (* check if there are no more lines that can be output
       on the present page *)

    if currentpageht = 0 then getnextpage;

  end;

end; (* processtuples *)
```

```
begin   (* processwords *)

      (*  obtain the first tuple from the relation INPUT *)
      writeln('enter processwords');
      rptr1:=findrel('INPUT');
      c := 'c' ;
      getuple(rptr1,c,ptr1,tptr1);
      (* from this tuple obtain the value fo the starting
         line number *)

      for indx1 := 1 to 8 do lnstr[indx1]:=' ';
      dummy := tptr1;
      for indx1:= (9 - lline) to 8 do
         begin
              lnstr[indx1]:=bufptr[ptr1]^.data[dummy+pline-1];
              dummy:=dummy+1;
              writeln;writeln('      line no =',lnstr:8);
         end;
      pageno:=1;
      charinttransform(lineno,lnstr);
      processtuples;
 end;   (* processwords   *)


Procedure printrels;
     var a,b : integer;


    (* THIS ROUTINE PRINTS OUT THE RELATIONS *)

    begin (* printrels *)
    writeln('enter printrels');
     writeln;writeln;writeln;
                 a:=findrel('INPUT');
                 rel[a].rindx:=0;
                 prtrel('INPUT','THE INPUT RELATION','');
                 writeln;writeln; writeln; writeln;
                 b:=findrel('TABLES');
                 rel[b].rindx:=0;
                 prtrel('TABLES','THE TABLES RELATION','');
                 writeln;
    end; (* printrels *)
```

APPENDIX B

```
procedure initialize;

   var n : integer;
       b : boolean;
       rdptr1,rdptr2 : index;
       domlst1,domlst2 : dlist;


  begin (* initialize *)

  (* INITIALIZE RELATIONS AS WELL AS ATTRIBUTE NAMES, SIZES, AND
     RELATIVE POSITIONS IN TUPLE *)

  domlst1[0]:='WORD';               domlst1[1]:='SEQ';
  domlst1[2]:='WORDLENG';           domlst1[3]:='LINE';
  domlst1[4]:='PAGE';

  domlst2[0]:='TNAME';              domlst2[1]:='TSIZE';
  domlst2[2]:='TWIDTH';             domlst2[3]:='TPAGE';
  domlst2[4]:='TFLAG';              domlst2[5]:='TRANK';

  n:=0;
  b:=findrd('INPUT',domlst1,n,rdptr1);
  n:=0;
  b:=findrd('TABLES',domlst2,n,rdptr2);

  pword:=rd[rdptr1[0]].pos;
  pseq:=rd[rdptr1[1]].pos;
  pwordlength:=rd[rdptr1[2]].pos;
  pline:=rd[rdptr1[3]].pos;
  ppage:=rd[rdptr1[4]].pos;

  ptname:=rd[rdptr2[0]].pos;
  ptsize:=rd[rdptr2[1]].pos;
  ptwidth:=rd[rdptr2[2]].pos;
  ptpage:=rd[rdptr2[3]].pos;
  ptflag:=rd[rdptr2[4]].pos;
  ptrank:=rd[rdptr2[5]].pos;

  lword:=pseq-pword;
  lseq:=pwordlength-pseq;
  lwordlength:=pline-pwordlength;
  lline:=ppage-pline;
  lpage:=4;

  ltname:=ptsize-ptname;
  ltsize:=ptwidth-ptsize;
  ltwidth:=ptpage-ptwidth;
  ltpage:=ptflag-ptpage;
  ltflag:=ptrank-ptflag;
  ltrank:=1;

end; (* initialize *)
```

```
Procedure TABLEPROGRAM;
     begin (* TABLEPROGRAM *)
         initialize;
         processwords;
         printrels;
     end; (* TABLEPROGRAM *)

begin (* T A B S I N T E X T *)
     setup('N','TDATA');
     TABLEPROGRAM;
     promptboot;
end.  (* T A B S I N T E X T *)
```

APPENDIX B

```
AAAAAA   PPPPP    PPPPP    EEEEEE   N      N   DDDD    IIIIIIIII  X       X
A    A   P    P   P    P   E        NN     N   D   D       I       X     X
A    A   P    P   P    P   E        N N    N   D    D      I        X X
AAAAAA   PPPPP    PPPPP    EEEE     N  N   N   D    D      I         X
A    A   P        P        E        N   N  N   D    D      I        X X
A    A   P        P        E        N    NN    D   D       I       X     X
A    A   P        P        EEEEEE   N      N   DDDD    IIIIIIIII  X       X

                                   CCCCCCC
                                  CCCCCCCCC
                                  CC
                                  CC
                                  CC
                                  CCCCCCCCC
                                   CCCCCC
```

```
(*$g++*)
(*$n+*)
Program SETUP_INPUT;
   uses globalerr screenops.syspro.sort.insavedump,
        numer1.cnsdomop.fss_util,actual.krunch.algebra.
        project1.select1.forms.tidvfcn.process1,editor.
        rel_ops.reconize,fss_hart,history;


   Procedure setup_data;
       var  domlist1.domlist2,tab1,tab2,tab3,tab4,tab5.
            tab6.rellist : dlist;
            domlen1.domlen2.tablen1,tablen2.tablen3.tablen4.
            tablen5.tablen6  : index;

       Begin

          domlist1[0]:='word';            domlen1[0]:=15:
          domlist1[1]:='seq':             domlen1[1]:=2;
          domlist1[2]:='wordleng';        domlen1[2]:=2;
          domlist1[3]:='line';            domlen1[3]:=2;
          domlist1[4]:='page':            domlen1[4]:=3;

          domlist2[0]:='tname';           domlen2[0]:=6;
          domlist2[1]:='tsize';           domlen2[1]:=2;
          domlist2[2]:='twidth';          domlen2[2]:=2;
          domlist2[3]:='tpage';           domlen2[3]:=3;
          domlist2[4]:='tflag';           domlen2[4]:=1;
          domlist2[5]:='trank';           domlen2[5]:=2;
```

APPENDIX C

```
tab1[0]:='order # ';          tablen1[0]:=2;
tab1[1]:='customer';          tablen1[1]:=30;
tab1[2]:='salesman';          tablen1[2]:=20;
tab1[3]:='assembly';          tablen1[3]:=10;
tab1[4]:='quantity';          tablen1[4]:=4;

tab2[0]:='order # ';          tablen2[0]:=2;
tab2[1]:='customer';          tablen2[1]:=30;
tab2[2]:='salesman';          tablen2[2]:=20;
tab2[3]:='assembly';          tablen2[3]:=10;
tab2[4]:='quantity';          tablen2[4]:=4;

tab3[0]:='student1';          tablen3[0]:=10;
tab3[1]:='student2';          tablen3[1]:=8;
tab3[2]:='asst';              tablen3[2]:=6;
tab3[3]:='exam';              tablen3[3]:=6;

tab4[0]:='student';           tablen4[0]:=10;
tab4[1]:='coursemk';          tablen4[1]:=8;

tab5[0]:='student';           tablen5[0]:=10;
tab5[1]:='course';            tablen5[1]:=8;

tab6[0]:='tname';             tablen6[0]:=6;
tab6[1]:='tsize';             tablen6[1]:=2;
tab6[2]:='twidth';            tablen6[2]:=2;
tab6[3]:='tpage';             tablen6[3]:=3;
tab6[4]:='tflag';             tablen6[4]:=1;
tab6[5]:='trank';             tablen6[5]:=2;

edit('',domlist1,domlen1,1,5,'INPUT',1,1);
edit('',domlist2,domlen2,1,6,'TABLES',1,1);
edit('',tab1,tablen1,1,5,'ORDER',1,1);
edit('',tab2,tablen2,1,5,'ORDER4',1,1);
edit('',tab3,tablen3,1,4,'MARK',1,1);
edit('',tab4,tablen4,1,2,'FINAL',1,1);
edit('',tab5,tablen5,1,2,'REGSTR',1,1);
edit('',tab6,tablen6,1,6,'TBLEX',1,1);
```

```
        rellist[0]:='INPUT';
        rellist[1]:='TABLES';
        rellist[2]:='ORDER';
        rellist[3]:='ORDER4';
        rellist[4]:='MARK';
        rellist[5]:='FINAL';
        rellist[6]:='REGSTR';
        rellist[7]:='TBLEX';

        prtrel('INPUT','THE INPUT RELATION','printer:');
        prtrel('TABLES','THE TABLES RELATION','printer:');
        prtrel('ORDER','TABLE 1 .... ORDER','printer:');
        prtrel('ORDER4','TABLE 2 .... ORDER4','printer:');
        prtrel('MARK','TABLE 3 .... MARK','printer:');
        prtrel('FINAL','TABLE 4 .... FINAL','printer:');
        prtrel('REGSTR','TABLE 5 .... REGISTER','printer:');
        prtrel('TBLEX','TABLE 6 .... TABLE --- Example','printer:');

        save(rellist,8);

    end; (* setup_data *)


Begin (* ex *)
   (*$r syspro*)
   setup('Y','TDATA ');
   setup_data;
End. (* ex *)
```

```
AAAAAA   PPPPP    PPPPP    EEEEEE  N      N  DDDD    IIIIIIIII X       X
A    A   P    P   P    P   E       NN     N  D   D       I      X     X
A    A   P    P   P    P   E       N N    N  D    D      I       X   X
AAAAAA   PPPPP    PPPPP    EEEE    N  N   N  D    D      I         X
A    A   P        P        E       N   N  N  D    D      I       X   X
A    A   P        P        E       N    NN   D   D       I      X     X
A    A   P        P        EEEEEE  N      N  DDDD    IIIIIIIII X       X
```

```
DDDDDDD
DDDDDDDDD
DD      DD
DD      DD
DD      DD
DDDDDDDDD
DDDDDDD
```

THE INPUT RELATION : BEFORE EXECUTION OF TABSINTEXT
RELATION : INPUT

word	seq	wordleng	line	page
The	01	03	01	00
text	02	04	01	00
is	03	02	01	00
as	04	02	01	00
follows:	05	08	01	00
The	01	03	02	000
final	02	05	02	000
array	03	05	02	000
represents	04	10	02	000
4	05	01	02	000
relation	06	08	02	000
which	07	05	02	000
is	08	02	02	000
said	01	04	03	000
to	02	02	03	000
be	03	02	03	000
a	04	01	03	000
projection	05	10	03	000
of	06	02	03	000
the	07	03	03	000
following	08	09	03	000
relation	09	08	03	000
Example:	01	08	04	000
Consider	02	08	04	000
the	03	03	04	000
relation	04	08	04	000
Order	05	05	04	000
*TORDER	00	00	05	000
A	01	01	06	000
permuted	02	08	06	000
projection	03	10	06	000
of	04	02	06	000
This	05	04	06	000
relation	06	08	06	000
is	07	02	06	000
as	08	02	06	000
follows	09	07	06	000
*TORDER4	00	00	07	000
The	01	03	08	000

APPENDIX D

following	02 09 08 000
example	03 07 08 000
is	04 02 08 000
an	05 02 08 000
indication	06 10 08 000
of	07 02 08 000
how	08 03 08 000
the	09 03 08 000
relation	01 08 09 000
"Tables"	02 08 09 000
is	03 02 09 000
derived	04 07 09 000
*TMARK	00 00 10 000
*TFINAL	00 00 11 000
*TREGSTR	00 00 12 000
The	01 03 13 000
resulting	02 09 13 000
"Tables"	03 08 13 000
relation	04 08 13 000
would	05 05 13 000
look	06 04 13 000
as	07 02 13 000
follows:	08 08 13 000
*TTBLEX	00 00 14 000

THE TABLES RELATION : BEFORE EXECUTION OF TABSINTEXT
RELATION : TABLES
tname tsize twidth tpage tflag trank

000000 00 00 000 0 00

TABLE 1 .... ORDER
RELATION : ORDER

order #	customer	salesman	assembly	quantity
04	Pennsylvania Railroad	Hannah Trainman	Car	37
03	London & Southwestern	Eric Brakeman	Car	23
02	New York Central	Natacha Engineer	Locomotive	47
07	Grand Trunk Railroad of Canada	Natacha Engineer	Locomotive	47
03	London & Southwestern	Eric Brakeman	Caboose	3
05	New York Central	Hannah Trainman	Locomotive	13
07	Grand Trunk Railroad of Canada	Natacha Engineer	Caboose	43
08	Great North of Scotland	Eric brakeman	Toy Train	37
01	Great North of Scotland	Eric Brakeman	Locomotive	2
05	New York Central	Hannah Trainman	Car	31
06	Baltimore & Ohio	Hannah Trainman	Car	17
04	Pennsylvania Railroad	Hannah Trainman	Toy Train	11
03	London & Southwestern	Eric Brakeman	Locomotive	5
01	Great North of Scotland	Eric Brakeman	Toy Train	7
07	Grand Trunk Railroad of Canada	Natacha Engineer	Car	137

TABLE 2 .... ORDER4
RELATION : ORDER4

order #	customer	salesman	assembly	quantity
04	Pennsylvania Railroad	Hannah Trainman	Car	37
04	Pennsylvania Railroad	Hannah Trainman	Toy Train	11

TABLE 3 .... MARK
RELATION : MARK

student1	student2	asst	exam
Brown	Brown	20.	50.
Hung	Hung	27.	58.
Jones	Jones	28.	62.
Raman	Raman	24.	66.
Smith	Smith	25.	60.

APPENDIX D

TABLE 4 .... FINAL
RELATION : FINAL

student	coursemk
Smith	85.
Jones	90.
Brown	70.
Hung	85.
Raman	90.

TABLE 5 .... REGISTER
RELATION : REGSTR

student	course
Brown	Aldat
Brown	Pascal
Hung	Algol68
Jones	Aldat
Jones	Algol68
Smith	APL
Smith	Pascal

TABLE 6 .... TABLE --- EXAMPLE
RELATION : TBLEX

tname	tsize	twidth	tpage	tflag	trank
ORDER	15	66	001	B	02
ORDER4	02	66	001	B	01
MARK	05	30	002	B	01
FINAL	05	18	004	B	01
REGSTR	07	18	004	T	02

```
AAAAAA  PPPPP   PPPPP   EEEEEE  N     N  DDDD   IIIIIIIII X     X
A    A  P    P  P    P  E       NN    N  D   D      I      X   X
A    A  P    P  P    P  E       N N   N  D    D     I       X X
AAAAAA  PPPPP   PPPPP   EEEE    N  N  N  D    D     I        X
A    A  P       P       E       N   N N  D    D     I       X X
A    A  P       P       E       N    NN  D   D      I      X   X
A    A  P       P       EEEEEE  N     N  DDDD   IIIIIIIII X     X

                           EEEEEEEEE
                           EE
                           EE
                           EEEEEEEEE
                           EE
                           EE
                           EEEEEEEEE
```

THE INPUT RELATION
RELATION : INPUT

word	seq	wordleng	line	page
The	01	03	01	1
text	02	04	01	1
is	03	02	01	1
as	04	02	01	1
follows:	05	08	01	1
The	01	03	02	1
final	02	05	02	1
array	03	05	02	1
represents	04	10	02	1
4	05	01	02	1
relation	06	08	02	1
which	07	05	02	1
is	08	02	02	1
said	01	04	03	1
to	02	02	03	1
be	03	02	03	1
a	04	01	03	1
projection	05	10	03	1
of	06	02	03	1
the	07	03	03	1
following	08	09	03	1
relation	09	08	03	1
Example:	01	08	04	1
Consider	02	08	04	1
the	03	03	04	1
relation	04	08	04	1
Order	05	05	04	1
*TORDER	00	00	05	1
A	01	01	06	1
permuted	02	08	06	1
projection	03	10	06	1
of	04	02	06	1
This	05	04	06	1
relation	06	08	06	1
is	07	02	06	1
as	08	02	06	1
follows	09	07	06	1
*TORDER4	00	00	07	1

APPENDIX E

The	01	03	08	1
following	02	09	08	2
example	03	07	08	2
is	04	02	08	2
an	05	02	08	2
indication	06	10	08	2
of	07	02	08	2
how	08	03	08	2
the	09	03	08	2
relation	01	08	09	2
"Tables"	02	08	09	2
is	03	02	09	2
derived	04	07	09	2
*TMARK	00	00	10	2
*TFINAL	00	00	11	2
*TREGSTR	00	00	12	2
The	01	03	13	2
resulting	02	09	13	2
"Tables"	03	08	13	2
relation	04	08	13	2
would	05	05	13	2
look	06	04	13	2
as	07	02	13	2
follows:	08	08	13	2
*TTBLEX	00	00	14	3

THE TABLES RELATION
RELATION : TABLES

tname	tsize	twidth	tpage	tflag	trank
000000	00	00	000	0	00
ORDER	15	66	1	b	1
ORDER4	2	66	1	b	2
MARK	5	30	2	b	1
FINAL	5	18	2	b	2
REGSTR	7	18	2	b	3
TBLEX	5	16	3	t	1

B I B L I O G R A P H Y

1. Chiu G., "*MRDSA User's Manual*", Technical Report SOCS 82.7, May 82.

2. Merrett T. H., "*Relational Information Systems*", Reston Publishing Inc, 1983.

3. Merrett T. H., Kazem Zaidi S. H.; "*MRDSP User's Manual*", Technical Report SOCS-81-27, Aug. 81.

4. Skillin, M. E., Gay R. M., "Words Into Type", 3rd Edition, Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1974.

5. Van Rossum T., "*Implementation of a Domain Algebra and a Functional Syntax*", Technical Report SOCS-83-18, Aug. 83.