NASA Technical Memorandum 78701

SPAR Data Handling Utilities

Gary L. Giles and Raphael T. Haftka

SEPTEMBER 1978

NASA

NASA Technical Memorandum 78701

SPAR Data Handling Utilities

Gary L. Giles
Langley Research Center
Hampton, Virginia
and
Raphael T. Haftka
Illinois Institute of Technology
Chicago, Illinois

National Aeronautics
and Space Administration

Scientific and Technical
Information Office

1978

Page intentionally left blank

Page intentionally left blank

CONTENTS

SUMMARY

The SPAR computer software system is a collection of processors that perform particular steps in the finite-element structural analysis procedure. The data generated by each processor are stored on a data base complex residing on an auxiliary storage device, and these data are then used by subsequent processors. The computer software associated with the data base complex provides a general capability and can be used for management of data in programs or systems other than SPAR.

This report documents the SPAR data handling utilities, which are routines used to transfer data between the processors and the data base complex. A detailed description of the data base complex organization is also presented. A discussion of how these SPAR data handling utilities are used in an application program to perform desired user functions is given with the steps necessary to convert an existing program to a SPAR processor by incorporating these utilities. Finally, a sample SPAR processor is included to illustrate the use of the data handling utilities. This information can be used (1) to understand more clearly and to use more productively the existing processors in the SPAR system, (2) to develop new SPAR processors, or (3) to use the capabilities of the data handling utilities in a system totally unrelated to SPAR.

INTRODUCTION

Large volumes of data for a variety of purposes are generated by finite-element structural analysis procedures. The organization of these data and the bookkeeping methods used to store and retrieve the data are important considerations in implementing the procedures as computer programs. Several large computer programs (e.g., NASTRAN® and ATLAS, refs. 1 and 2) contain methods to perform these data handling functions. Such analysis programs are often used in more comprehensive systems, as discussed in references 3 and 4. The data content, organization, and handling methods must be carefully defined to provide for communication of data between programs.

The SPAR finite-element structural analysis system (ref. 5) contains a data base complex with a standardized organization of data and a self-contained set of data handling utilities used to access the data. These data handling utilities were developed by W. D. Whetstone and provide a key capability which contributes to the effectiveness of the SPAR analysis system. The computer software associated with the data handling utilities and data base complex is a general capability and can be used for management of data in programs or systems other than SPAR. Data handling functions that can be performed by a SPAR user are described in the SPAR reference manual (ref. 5).

The purpose of the present report is to extend the available documentation of the SPAR system by providing a detailed description of the data base complex

organization and the data handling utilities used to communicate with the data base. A general overview of the technical capabilities and organization of the SPAR system is given in the beginning sections of the report. Then the contents and organization of the data base complex are described. A discussion of how the data handling utilities perform desired user functions in an application program is given with detailed descriptions of each of the data handling routines provided in appendixes A to D. A step-by-step procedure to incorporate these routines into existing programs is given. Finally, a sample program which contains these routines is included in appendixes E and F to illustrate the use of the capabilities described herein. This information can be applied to use the SPAR system more productively, to make additions or modifications to the existing system, or to use the SPAR data handling utilities and associated programing practices in computer programs totally unrelated to SPAR.

Use of commercial products and names of manufacturers in this report does not constitute an official endorsement of such products or manufacturers, either expressed or implied, by the National Aeronautics and Space Administration.

SPAR TECHNICAL CAPABILITIES

SPAR is an analysis system capable of computing static deflections and stresses, natural vibration frequencies and modes, and buckling loads and mode shapes of linear finite-element structural simulations. The structural simulations are composed of finite elements connected at specified joints, which can have three translational and three rotational components of deflection. Finite elements which are currently available for simulating the stiffness characteristics of a structure include axial bars, beams of general cross section, triangular and quadrilateral plates having an option to specify coupled or uncoupled membrane and bending stiffness, and quadrilateral shear panels. Properties of the plates may be specified as layers in a laminate of composite materials, and there is provision for warping of the quadrilateral plate element. Mass properties of a structure are represented by structural and nonstructural masses associated with the stiffness elements and by concentrated masses at the joints. Loading data can include any or all of the following categories: point forces or moments acting at the joints, specified joint motions, inertial loading, thermal or pressure loads, and initial strains in individual elements. This technical capability contained in the SPAR system is available for operation on Control Data (CDC) 6000 or CYBER 170 Series, UNIVAC 1100 series (ref. 5), or Prime 400 computer systems (ref. 6).

The finite-element structural analysis procedure is divided into a sequence of steps or functions which must be performed. The computer code required for each of these steps is referred to herein as a "processor." Processors are separate portions of the SPAR system which are selectively executed in a logical sequence to perform a desired analysis. Each processor is designed to perform a limited, yet distinct and complete, function. The functions of each of the processors in the version of SPAR denoted as level 11 are given in table 1. The processors TAB through KG read user input, from element matrices, and assemble element matrices into system matrices which represent the overall stiffness and mass of the structure. Solutions of the system matrix equations are performed by INV through PSF. The EIG and DR processors are used for eigensolu-

tions and dynamics response analyses, respectively. Calculations involving substructuring are performed by SYN and STRP. The remaining processors listed in table 1 provide three types of functions which are general and not limited to structural analysis: (1) the arithmetic utility system (AUS) provides general matrix input and arithmetic capability, (2) the data complex utilities (DCU and VPRT) manage and print data, and (3) PLTA and PLTB processors perform on-line and off-line plotting.

A general characteristic of the SPAR processors is their efficiency with respect to both computer memory and processing time requirements. The method for handling the large, sparse matrices encountered in finite-element structural analysis in a manner to achieve this efficiency is described in reference 7. All the processors make extensive use of auxiliary disk storage and operate with computer memory which is automatically or dynamically allocated so that large structural simulations can be analyzed. The method used for eigenvalue and eigenvector calculations (ref. 8) performs a vibration analysis without first reducing the number of degrees of freedom being considered. This capability allows a single simulation to be used for both static and dynamic calculations.

The SPAR system is designed for effective interactive operation via teletype and/or graphics terminals. All input is in free-field format, and executive control commands call selected processors for execution. The related input data for a given processor is typed in sequentially at a user console keyboard after successive prompts from the operating system. The computational sequence is continued by using another executive control command to call the next desired processor. This flexibility for selective execution of various processors gives the user considerable versatility in performing structural analyses.

SPAR SYSTEM ORGANIZATION

The organization of the various portions of the SPAR analysis system is shown schematically in figure 1. User input to the system is indicated with the executive control commands (@XQT followed by the name of the processor to be executed) and related input data located sequentially on the input file. This input file, shown at the upper left of the figure, is a standard input file for UNIVAC systems, and the CDC and Prime versions of SPAR are also designed to accept input in nearly identical format.

Names designating each of the processors illustrate the portion of the system which performs all the analytical computations. Only a single one of these processors is in central memory of the computer at any time during an analysis run. For UNIVAC and Prime computers, the processors are separate programs which are called sequentially by execution commands to the operating system (e.g., EXEC-8). For the CDC version of SPAR, all processors are designated as primary overlays in a single program which has a zero level overlay to call the processors for execution.

Each processor has a working storage area (shown as a single block applicable to all processors in fig. 1) which contains most of the data that are input to, calculated by, and/or output from the processor. This working storage area is implemented as a single large vector located in blank COMMON of the processor.

Numerical values contained in various arrays used by the analysis subroutines are stacked in this area in vector form. The lengths of these arrays are, in general, problem dependent and the starting address for each array within the large vector is calculated internally by each processor. This dynamic storage allocation feature makes efficient use of the central memory of the computer.

A set of data handling utilities (the same for each processor) is used to transfer data between the working storage area of the processor in central memory of the computer and a group of files located on an auxiliary storage device, as shown in figure 1. This group of files is referred to as the data base complex. Herein the collective term "routines" refers to both the subroutines and the functions which comprise the data handling utilities and are called by the processors to perform a variety of data handling tasks. These utilities perform all data transfer between processors, since data from one processor are written on the data base complex and data needed for the next processor are read from the data base complex. Data transfer takes place directly between a specified location in the working storage area and a specified location on disk, without the intermediate, and costly (in both storage and processing time), step of going through a buffer area in central memory. These data handling utilities are independent of the type of analyses performed by the processors. A discussion of how these utilities communicate with the data complex is given in a subsequent section of this report, and detailed descriptions of each of the routines are given in the appendixes.

The data base complex is composed of data files resident on auxiliary disk or drum storage, as shown at the bottom of figure 1. A maximum of 26 of these data files, each referred to as a "library," are available for use. Users can elect to store the entire data complex in a single library file. These files are recognized by the computer operating system as having names SPARLA . . . SPARLZ, as shown in figure 1, for Control Data versions; SPAR-A . . . SPAR-Z for UNIVAC; and SPLA . . . SPLZ for Prime. The files are referred to by the SPAR user as libraries 1 to 26, with libraries 1 to 20 available for general use and libraries 21 to 26 reserved for temporary internal use. These library files can be retained by the user at the end of a run and the information used in subsequent runs. This data retention is accomplished without data reformatting procedures, and the user need not be concerned with the internal structure of the data complex. All libraries contain data in the form of data sets and have an identical organization. A description of this library organization is given in the next section.

DESCRIPTION OF THE SPAR DATA BASE COMPLEX

The SPAR data base complex is located on auxiliary disk or drum storage devices. These devices are divided into sectors (containing 28 words on UNIVAC drums, 55 words on Prime disks, and 64 words on CDC disks), and each read and write operation must begin at the beginning of a sector. Therefore, the location of all data in the data base complex is given in terms of a library number and a relative sector number within that library. The contents of a library file in the data base complex are described in this section, and the data handling routines used to create and access the data are described in a subsequent section. Information concerning the contents of the library is similar to that

contained in the SPAR reference manual (ref. 5) but is included herein because it serves as a necessary background to the understanding of the data handling routines.

Library Structure

A set of data in the SPAR data base complex is referred to by giving the library number and a unique name assigned to that data set. A procedure to relate the data set names to a corresponding relative sector location on disk is provided by the data handling utilities. Information on a SPAR library includes tables necessary to locate the analytical data in addition to the analytical data itself. The structure and contents of this information are shown schematically in figure 2. The structure is a three-level hierarchy. The top level is a master directory which contains the locations of a set of tables of data set names and their corresponding locations. These tables are collectively referred to as a table of contents (TOC), which is separated into as many as 61 segments, as indicated in figure 2. Each of the segments in a table of contents contains information to describe and locate up to 32 data sets (shown as the bottom level for only the first segment), which contain the analytical data values. This structure of the master directory, table of contents, and data sets on the same library makes each library a self-contained entity independent of the processors and the other libraries.

When a user wants to read a data set from the data complex, the library number and the data set name are all the information that is needed. The segments of the TOC for the specified library are searched automatically until a match with the data set name is found. The disk address and size of the specified data set are contained in the TOC line, and that information is used in reading the data into central memory.

The contents of a library are stored in sequential relative sectors of the library file. The master directory begins at relative sector zero followed by the first segment of the table of contents and then its corresponding data sets. Subsequent segments of the table of contents and data sets are repeated sequentially to the end of the library. Therefore, a maximum of $61 \times 32 = 1952$ individually named data sets can be stored on a single library file. Utilities exist in SPAR to store complete libraries as a single data set inside another library (nesting of libraries) so that a user can organize data in a heirarchy of libraries if desired. These nested libraries must be reconstituted into separate libraries before the individual data sets can be accessed.

The overall structure of a library has been described starting at the top level shown in figure 2. A detailed description of the contents of each of the levels follows, starting at the bottom level with data sets, then the table of contents, and finally the master directory.

The Data Set

The data in a library are stored as a series of computer words. A SPAR data set is a grouping of one or more words which are stored in a library and

which can be referred to or accessed as a single entity. The contents and organization of the words within a data set are determined by the user or programer who originates the data set. All data that are communicated among SPAR processors are handled in terms of data sets.

The data sets are identified by a four-word name: NAME1, NAME2, NAME3, and NAME4. The words NAME1 and NAME2 contain up to four alphanumeric characters, and NAME3 and NAME4 are integers. The following are examples of valid names:

JLOC	BTAB	2	5
K	SPAR	36	0
VIBR	MODE	1	1

The names may be used to indicate the contents of the data sets; for example, the first example above is used to refer to joint locations; the second, to the assembled structural stiffness matrix; and the third, to natural vibration mode shapes.

A data set may contain a large number of words, and situations may arise when it is desirable to read or write only a few of these words at a time. For example, when handling very large data sets, such as the assembled stiffness matrix, it is often desirable to read or write the data set after it has been broken down into a series of smaller blocks. Each block within the data set is stored beginning at a new disk sector so that it can be accessed individually.

A standard form is used for all data sets contained in the libraries. Each data set is composed of a number of blocks, and each block may be interpreted as a two-dimensional matrix, as illustrated in figure 3. These matrices are dimensioned (NI,NJ), and each block length is always NI times NJ. The information within each block is ordered by column when stored on the disk. Each block begins at a new sector. If the block length is not an integral multiple of the sector length, some unused disk storage results at the end of each block, as illustrated in the figure. The example shown in figure 3 could refer to vibration mode shapes with six degrees of freedom at seven joint locations. Each new mode shape would start at the beginning of a new sector and could be accessed individually.

Table of Contents

A table of contents (TOC) is used to store and relate the names, disk addresses, and characteristics (size, type, etc.) of all data sets resident in the data base complex. A TOC exists for each library that is used. A listing of a typical TOC is shown in table 2. Such a listing is produced by a SPAR utility processor and provides a concise summary of the data produced by a given computer run.

A TOC listing contains one line or row per data set with each line containing 12 entries, as shown in table 2. The first column, denoted SEQ, provides a sequence number for each of the data sets and is not actually stored as one of

the 12 entries. The four-word data set name (shown as N1, N2, N3, and N4 in table 2) is given as the last four entries and is used to identify a particular data set. A description of the first eight entries for a data set are given in the following table (from ref. 5):

TOC item	Description
RR	Disk address pointer to first word of data set at beginning of a new sector; a preceding minus sign means that the data set has been "disabled" (still resident on disk but cannot be accessed)
DATE	Date of insertion
TIME	Time of entry into the processor which inserted the data set into the library
ER	Error code: 0 no error detected during generation of the data set 1 minor error 2 fatal error -1 incomplete data set
WORDS	Total number of words in the data set; data sets are generally comprised of a sequence of physical records, or "blocks"; each block is a two-dimensional matrix dimensioned (NI,NJ), i.e., NI rows, NJ columns; the block length is always NI*NJ
NJ	See above
NI*NJ	See above
TY	Type code: 0 integer -1 real -2 double precision 4 alphanumeric

An understanding of the information contained in a TOC is necessary for using the data handling routines in a user program to access the data sets. Several TOC entries are used as parameters or arguments which are passed to the routines at execution time. When TOC information is needed in central memory, it is stored in COMMON block CLIB, as described in appendix D.

Master Directory

A SPAR library begins at sector zero with MASTER, a 64-word array, which is the master directory for that library. This array contains the following information:

MASTER(1) Number of the first unused sector in the library (the upper end of the file)

MASTER(2) Number of TOC segments currently in the library

MASTER(3) Number of data sets per segment (presently fixed at 32)

MASTER(3+N) Sector number of beginning of TOC for Nth segment

Since MASTER is a 64-word array, the maximum number of segments that a library may contain is 61. The values of MASTER(1) and MASTER(2) are updated as data are added to the library.

As indicated, each library is self-contained, with MASTER containing pointers to TOC's for each segment and each TOC segment containing pointers to the data sets. Utilities exist in SPAR to store complete libraries as a data set (nesting of libraries), so that an unlimited number of data sets can be stored on a single library if desired.

Any data sets to be added to the library are written at the upper end of the file at the relative sector number given in MASTER(1). If a data set with the same name as the new one already exists on the library, the old one is disabled (indicated in the TOC by a minus sign appended to the relative sector number). When master directory information is needed in central memory, it is stored in COMMON block CLIB, as described in appendix D.

DESCRIPTION OF SPAR DATA HANDLING UTILITIES

There are two basic ways of communicating with the SPAR data complex. The first is to use the utility processors included in SPAR, and the second is to use the SPAR data handling routines directly to create new processors.

The first method is useful for performing general utility operations such as entering data created by other programs into the SPAR data complex, printing data sets, or moving data sets between libraries.

The data complex utility (DCU) provides the capability to print tables of contents as well as individual data sets. A COPY command is provided to transfer specified data sets between libraries. This capability is very useful in saving selected information between runs. The XCOPY and the XLOAD command are used to copy data sets from the direct-access libraries to sequential files and vice versa. These sequential files can then be used to interface other programs with the SPAR system or can be stored on magnetic tape for future use. Commands are also available in DCU for disabling and enabling or changing names of data sets and for nesting and reconstituting SPAR libraries. The arithmetic utility system (AUS) provides for input of user-defined data sets into the library, as well as general matrix arithmetic operations with any of the data sets. This mode of working with data sets in the data complex is described in the SPAR reference manual (ref. 5) and provides the flexibility to use the SPAR system in "nonstandard" applications.

The second method involves embedding statements to call the data handling routines directly in the user program. This method is desirable for the programer who is writing new SPAR processors, modifying existing ones, or possibly using the data complex to handle data in a system which is totally unrelated to SPAR. An overview of available routines and the functions they perform in a user program is given in this section. Detailed descriptions of these routines are given in appendixes A and B.

Organization of Data Handling Utilities

The purpose of the data handling utilities is to perform all data communication between the SPAR libraries on disk storage and the user processors in central memory. The relationships of the routines used for this purpose are shown schematically in figure 4. The directions of the arrows between routines in the figure indicate that the routine at the tail of the arrow is called by the routine at the head of the arrow to perform some desired function.

A user processor normally makes direct use of only the five routines (RIO, DAL, LTOC, TOCO, and FIN) shown at the top of figure 4. The routines RIO and DAL transfer (read and write) the contents of the data sets between disk and central memory. The routines LTOC and TOCO retrieve information from segments of the table of contents. This TOC information is often needed in calls to RIO and DAL. Finally, FIN closes all files at the end of a successful exit from a user program or processor or aborts a run after a fatal error is discovered.

Reading or writing on the disk is performed by the routine WR. In the UNIVAC version, WR is an assembler routine, whereas in the CDC version it is a FORTRAN routine that calls six COMPASS routines which were adapted from the NASTRAN program (ref. 1). The Prime version has routines written in the FORTRAN language to perform the reading and writing. The SPAR utility routine STATIO calls such routines for each computer system to perform the function of opening, rewinding, closing, and reading information into the file environment table (FET) for each file in the library. Other routines are called by RIO through WR to perform all the read and write operations directly between disk and central memory.

The four routines (MATCH, NTOC, RDIND, and WRTIND) shown in the center of figure 4 perform the functions necessary to store, interrogate, and retrieve information from the table of contents. To make efficient use of central memory, only the master directory and a single segment of the table of contents for each of two libraries are resident in central memory at any one time. Therefore, the master directory and table of contents information, required to locate a specified data set, must be read into central memory from disk and, if changed, must be written back onto disk before a new set of information is requested. The NTOC routine contains logic for performing these exchanges and calls RDIND and WRTIND for reading and writing of the table of contents segments. The MATCH routine searches successive segments of a table of contents until a match with a set of four specified data set names is found.

The use of the routines RIO, DAL, LTOC, TOCO, and FIN is discussed in the next section, and detailed descriptions are given in appendix A. Detailed

descriptions of the other utility routines shown in figure 4 are given in appendix B.

Functions Performed by Data Handling Utilities

In this section a description is given for using RIO, DAL, LTOC, TOCO, and FIN to perform desired functions within a user program or processor. A general discussion of these routines is given by user function, and the formal parameters used to call the routines are described in appendix A.

Creating data sets.- Data sets that contain only one block may be created with a single call to DAL. For example,

```
CALL DAL(1,1,KA,0,1,KADR,IERR,88,1,88,0,4HCUCU,4HCUCU,1,0)
```

creates a data set with the four-word name CUCU CUCU 1 0 in library 1 and writes into it the first 88 words of the array KA. On return KADR contains the first sector number for this data set.

If a data set contains more than one block, it may be created by a call to DAL to open the data set, followed by several calls to RIO to write out the individual blocks of the data set. For example, to create a data set containing the array TRID(5,10,7) such that the last index is the block number, the following sequence of FORTRAN statements is used:

```
      CALL DAL(NU,0,TRID,KORE,IEA,KADR,IERR,350,10,50,-1,4HTRID,4HARRY,0,7,)
      DO 100 I=1,7
100   CALL RIO(NU,10,2,KADR,TRID(1,1,I),50)
```

The first call to DAL opens a data set called TRID ARRY 0 7 and specifies its parameters (number of words, block size, etc.); KADR is returned with the sector number where the data set is to be written. Note that DAL is used with IOP=0 (see DAL description in appendix A for definition of IOP), so that not a single word of the data set is actually written into storage. The seven calls to RIO transfer the seven blocks of TRID into storage.

Sometimes the number of blocks or the block size that is to be written on a data set is not known ahead of time. In this case any numbers (or zeros) may be used for these parameters in the initial call to DAL to open the data set. Later after all the blocks of that data set have been written into storage, another call to DAL (CALL DAL(NU,-1...)) is used to change the parameters of the TOC for the data set. Care should be taken to avoid calling DAL to open or write another data set into the same library before the first data set has been completely written by appropriate calls to RIO.

Reading data sets.- The first block in a data set may be read by calling DAL with IOP=11. For example, the statement

```
CALL DAL(1,11,KA,KORE,IEA,KADR,IERR,NWDS,NE,LB,ITYPE,4HCUCU,4HCUCU,1,0)
```

reads into central memory at beginning location KA the first block of the data set CUCU CUCU 1 0. Additionally, the data set parameters NWDS, NE, LB, and ITYPE are returned and KADR is set to be the first sector number of the data set. If several blocks of a data set are to be read starting from the first one, the sector number of the first block in the data set must be found and then RIO used to read the data set block by block. This information may be obtained by using the subroutines DAL or TOCO or the function LTOC. For example, the first sector number of data set TRID ARRY 0 7 is placed in KADR by either of the following three sequences:

```
 (i) KADR = LTOC(NU,1,4HTRID,4HARRY,0,7)

(ii) CALL DAL(NU,10,KA,KORE,IEA,KADR,IERR,NWDS,NE,LB,ITYPE,4HTRID,4HARRY,0,7)

(iii) COMMON/TOCLIN/LINE(12)
      NA4(1) = 4HTRID
      NA4(2) = 4HARRY
      NA4(3) = 0
      NA4(4) = 7
      NLINE = 0
      CALL TOCO(NU,NA4,1,NLINE)
      KADR = LINE(1)
```

In (i), only the desired single parameter KADR is obtained. In (ii), the other data set parameters (NWDS, NE, LB, and ITYPE) are also returned. The third sequence (iii), using TOCO, gives the most extensive information, as all 12 TOC entries are placed in COMMON/TOCLIN/LINE(12).

Once the sector number for the first block is obtained, RIO may be used to read several blocks. For example, to read 7 blocks each of 50 words into the array TRID(5,10,7) starting at sector number KADR, the following sequence is used:

```
      DO 100 I=1,7
100 CALL RIO(NU,20,2,KADR,TRID(1,1,I),50)
```

Reading or modifying a part of a data set.- At times, it is desirable to read or modify only a few words of a data set rather than the entire data set. This action is particularly important when the data set is very large. It is relatively simple to read or modify an entire block of a data set; it is slightly more complicated to read or modify part of a block.

To read or modify one block of a data set, the sector number of the first word of the block (each block begins at the beginning of a sector) must be located. The first step is to locate the sector number KADR where the data set starts. This may be done by using subroutines DAL or TOCO or function LTOC. (See example in section "Reading data sets.") The second step is to calculate the sector number of the block. If the Nth block is needed and the block size is LB words (LB is obtained with a procedure similar to that for obtaining KADR, as discussed in the previous section), then the sector number of the block is obtained by the FORTRAN statement

```
KSHFT = (N-1)*NSECTS(LB)+KADR
```

where the SPAR function NSECTS returns the number of sectors required for LB words. A call to RIO may now be used to read or write this block.

If a block is composed of several sectors, it may be desired to read or modify one sector of the block. For a data set beginning at sector KADR (see previous subsection for information on obtaining KADR) and having a block size of LB, the sector number KSHFT containing the Mth word of the Nth block is obtained by the FORTRAN statement

```
KSHFT = KADR+(N-1)*NSECTS(LB)+NSECTS(M)-1
```

The desired word is the Lth word in the sector, where L is obtained by

```
L = M-(NSECTS(M)-1)*LSECT
```

and the sector length LSECT resides in COMMON/CINDEX/INDEX(7),LSECT (where LSECT = 28, 55, and 64 for UNIVAC, Prime, and CDC, respectively). Finally, RIO is used to read the sector into central memory, say into array KA, with the following statement:

```
CALL RIO(NU,2,2,KSHFT,KA,LSECT)
```

If RIO is used to change a data set, it is recommended practice to call DAL(NU,-1,...). This call changes the date and the time in the TOC for the altered data set and alerts the user that an alteration has been made.

Closing data files.- The last operation in each user program or processor is to close the files used by the processor. This operation is performed by using CALL FIN(0,0). Calling this routine insures that the library files which are generated by a processor are accessible to the subsequent processors.

Detailed descriptions of routines.- The definitions of formal parameters used in statements to call data handling routines which interface with user programs are given in appendix A. Similar descriptions of routines to perform data handling support functions and to initialize SPAR processors and read information from the input file are presented in appendixes B and C, respectively. Data (in addition to formal parameters) are also communicated between these routines through labeled COMMON blocks, and the contents of these blocks are defined in appendix D.

The routines described in appendixes A to C form a self-contained software package that can be used effectively in any engineering applications program or system. Examples of use of this SPAR software can be found throughout the listings of the SPAR processors.

GUIDE FOR IMPLEMENTING SPAR DATA HANDLING UTILITIES

Many computer programs presently exist and new ones are being developed to perform a variety of engineering analysis and design tasks. One approach

to producing a better product using this capability is to collect all applicable computer programs into a software system that can be used for extensive analysis of that product. Such software systems require communication of data among the various programs, and this section describes the steps necessary to convert existing programs so that they can use the SPAR data handling utilities for this purpose. Similar logical steps could be used in designing new programs which are intended to use this data management capability. The steps are

(1) Divide the existing program or process into the lowest level group of functions that the user might be interested in performing. It is important that this division be made on a functional rather than a computer programing basis. An example of how the finite-element structural analysis procedure was divided into processors is given in table 1. The computer code required for each of these functions is referred to as a module or processor. These processors should be designed to operate as independent programs for UNIVAC and Prime versions or as primary overlays on CDC versions. All data transfer should be handled through the data base complex. The zero-level overlay for the new system would have to be able to recognize the executive control command ([XQT on CDC and Prime and @XQT on UNIVAC) for the name of each processor and call it into central memory for execution.

(2) Convert the processor to use dynamic allocation of blank COMMON area in central memory for working storage for all problem-size dependent arrays. This step is not necessary but is very desirable to provide efficient use of central memory. An example of dynamic storage allocation is shown in appendix G. Conversion of an existing program generally requires a new subroutine to set up starting addresses for arrays in blank COMMON and to call existing subroutines using these blank COMMON addresses in the calling sequences. The other subroutines that are called must be altered to provide the proper respective formal parameters in the subroutine statement and to provide dimension statements indicating that these parameters are array names. This programing technique can result in considerable savings of computer resources compared with programs containing arrays with fixed dimensions.

(3) Identify all input and output data for the program or processor. Input data include both user-supplied input for the processor and data sets which have been generated by other processors and are available for use from the data base complex. Output data refer to named data sets which are to be written on the data libraries so they can be used by subsequent processors. No special considerations are required for printed output. Handling of user-supplied data is discussed in steps (6) and (7), and input and output of data sets to and from the data base complex are discussed in steps (4) and (5). Four-word names must be assigned to each of the data sets to be written onto the data base complex.

(4) Examine size characteristics of each data set to determine whether the set can be handled most efficiently in a single block or in multiple blocks.

(5) Insert statements into the processor to call the appropriate data handling routines. As mentioned in the section describing functions performed by the data handling routines, LTOC and TOCO are often used to get size information for existing data sets for use in allocating blank COMMON storage. The DAL rou-

tine is used to handle data sets contained in a single block. A combination of DAL and RIO is used to handle data sets with multiple blocks.

(6) Provide a call to the RSET routine (see appendix C) at the beginning of each processor. A DATA statement must be set up which specifies the name, type (integer, real, or alphanumeric), and default value of each parameter to be used in a RESET statement for the processor. RESET parameters are used to provide an option for the user to specify parameter values which control execution of the processor, such as the library numbers or names of input and/or output data sets, scale factors, iteration controls, and selection of case or condition numbers. The routine RSET causes calculation and printing of the current amount of blank COMMON available in central memory (which is a function of user-specified field length for CDC versions). This value can be compared with that dynamically allocated for required use by each processor.

(7) Replace all statements which read data from the input file with calls to the READER routine. This routine provides a standard method of reading all input records in a free-field format. Use of a free-field format is particularly desirable when entering data from an interactive terminal and simplifies user documentation for a processor. Calls to the READER routine can be used to read tables or vectors and arrays of user input data; however, an alternate method is recommended. The alternate method is to use the data set construction capability of the AUS utility of SPAR to insert the required data sets into the data base complex before calling the processor which uses them. For example, this method is used in the present SPAR system to input data sets containing applied-loading information.

(8) Provide a call to the FIN routine at the end of each processor to close all files which are used. Routine FIN also causes printing of the time on the clock measuring CPU time for the job as well as a cumulative count of times data were written onto disk or read into central memory during execution of a processor. This information provides the user some measure of performance for each processor for the application being made.

(9) Compile and load the source of the processors to form executable files. For UNIVAC and Prime versions, each processor is a separate executable file. For CDC versions, the processors can be grouped into overlaid programs (not necessarily a single overlaid program). Since the processors all communicate through the data base complex, they can be called in the desired sequence to perform a particular set of calculations.

(10) Provide user documentation for the processor. User documentation is written as a self-contained section for each processor. The information in each section includes (a) the function of the processor, (b) a description of the RESET parameters, (c) names of input and output data sets for the processor, (d) central memory requirements (blank COMMON working area) given in terms of variables that are generally problem size dependent, and (e) code release information giving version of program, date coded, and originator of code.

CONCLUDING REMARKS

The SPAR computer software system is a collection of processors that perform particular steps in the finite-element structural analysis procedure. The data generated by each processor are stored on a data base complex residing on an auxiliary storage device, and these data are then used by subsequent processors. The organization of the data base complex provides significant benefits to the user, such as reference to data by alphanumeric names and automatic "bookkeeping" procedures for the data; expedites communication of data with programs external to SPAR; and simplifies retention of data between separate computer runs. The computer software associated with the data base complex provides a general capability and can be used for management of data in programs or systems other than SPAR.

This report documents the SPAR data handling utilities, which are routines used to transfer data between the processors and the data base complex. A detailed description of the data base complex organization is also presented. A discussion of how these SPAR data handling utilities are used in an application program to perform desired user functions is given with the steps necessary to convert an existing program to a SPAR processor by incorporating these utilities. Finally, a sample SPAR processor is included to illustrate the use of the data handling utilities. This information can be used (1) to understand more clearly and to use more productively the existing processors in the SPAR systems, (2) to develop new SPAR processors, or (3) to use the capabilities of the data handling utilities in a system totally unrelated to SPAR.

Langley Research Center
National Aeronautics and Space Administration
Hampton, VA 23665
June 19, 1978

APPENDIX A

SPAR DATA HANDLING ROUTINES CALLED BY USER PROGRAM

There are primarily five data handling routines that are needed in user programs to communicate with the data base complex. The RIO routine is the basic input/output routine in this group and handles data sets with multiple blocks or reads or writes information within a block. Routine DAL is used to open new data sets, to read and write data sets contained in a single block, and to modify TOC entries. Routines LTOC and TOCO are used to read information from the table of contents. Routine FIN is used to close all files at the end of a processor. A description of the formal parameters used for each of these routines (identified as either a subroutine or a function) is given in this appendix. In addition, the information contained in labeled COMMON blocks to communicate information between these routines is given in appendix C.

Subroutine RIO(NU,IWR,IOP,KSHFT,KA,L)

Subroutine RIO reads or writes L words from or into library NU.

IWR — Operation code (user specified):

1	write
2	read
10	write and return next sector number in KSHFT
20	read and return next sector number in KSHFT

IOP — Disk location code (user specified):

1	reads or writes at the disk sector number defined by adding KSHFT to last used sector number
2	reads or writes at sector number KSHFT
3	writes at first unused sector

KSHFT — Sector number or shift - see IOP (user specified and returned)

KA — Starting central memory address for I/O (user specified)

L — Number of words to be written or read (user specified)

Subroutine DAL(NU,IOP,KA,KORE,IEA,KADR,IERR,NWDS,NE,LB,

ITYPE,NAME1,NAME2,NAME3,NAME4)

Subroutine DAL reads or writes a data set identified by the name NAME1, NAME2, NAME3, NAME4, from or into library NU. Additionally, DAL can initialize, change, or retrieve some of the parameters defining the data set. The name of the data set is composed of two alphanumeric words having up to four characters (NAME1 and NAME2) and two integers (NAME3 and NAME4). If DAL is used to operate on an existing data set, one or more of these names may be masked (i.e., set to

4HMASK). Subroutine DAL would use the first active data set it encounters that agrees with the unmasked parts of the given name.

NU — Library number (user specified)

IOP — Operation code (user specified):

- -1 change the parameters defining the data set in TOC according to the parameters in the calling sequence and current date and time; IERR is set if data set not found; the disk address pointer (KADR) is set at the first sector number for the data set
- 0 set up an entry (a line) in TOC for a new data set, and disable any old duplicate (same name) data sets; disk address pointer (KADR) is set to where the data set will be written (but data set is not actually written)
- 1 same as IOP=0 but a data set is also written into disk at the end of the existing sets; the data set is written as one block, even if LB is different from NWDS; subroutine DAL should not be used to write data sets composed of more than one block
- 2 same as IOP=1 except old data sets having the same name are not disabled
- 10 get information from TOC without reading data set; IERR is set if data set is not found
- 11 same as IOP=10 but one block (LB words) of a data set is also read into central memory

KA — Starting address in central memory of data set (user specified)

KORE — Available central memory for data set (user specified and this value is available as the first word of blank COMMON after RSET is called); if LB is larger than KORE and IOP≥1, then IERR is set to -2; if KORE=0 the check for available central memory is skipped

IEA — Error return indicator (user specified):

- 1 print error message and return
- 2 disregard error
- Otherwise print error message and abort

KADR — First sector number of data set on disk (returned)

IERR — Error code (user specified for write, returned for read):

- 0 no errors
- -1 data set not found
- -2 insufficient central memory

NWDS — Number of words in data set (user specified for write, returned for read)

NE — Number of columns per block (user specified for write, returned for read)

LB — Block size in words (user specified for write, returned for read)

ITYPE — Type of data (user specified for write, returned for read):

0	integer
-1	real
-2	double precision
4	alphanumeric

NAME1 — Four-character alphanumeric word (user specified)

NAME2 — Four-character alphanumeric word (user specified)

NAME3 — Integer (user specified)

NAME4 — Integer (user specified)

Function LTOC(NU,J,NAME1,NAME2,NAME3,NAME4)

Function LTOC retrieves the Jth item from the TOC line for data set NAME1, NAME2, NAME3, NAME4 in library NU. If the data set is not found, then LTOC is equal to 4HXXXX on return.

NU — Library number (user specified)

J — Item number in TOC line; for example RR = 1, DATE = 2 (user specified)

NAME1...NAME4 — Data set name (user specified)

Subroutine TOCO(NU,NA4,IOP,NLINE)

This subroutine searches for a data set named NA4(1), NA4(2), NA4(3), NA4(4) in library NU. One or more of the entries in NA4 may be masked (i.e., set to 4HMASK) and those entries are disregarded in matching NA4 against the data set names. The search starts at line number NLINE in the table of contents.

NU — Library number (user specified)

NA4 — Four-word array containing the data set name (user specified)

IOP — Operation code (user specified):

1 — TOCO retrieves the first matching data set it encounters; the TOC entry of the data set is stored in COMMON/TOCLIN/LINE(12) and NLINE is set to be the line number of the data set; if the data set is not found, then NLINE is set equal to -1

≠ 1 — TOCO deactivates all data sets matching NA4 found after line NLINE; on return NLINE is equal to the number of data sets that were deactivated

NLINE Line number in the table of contents (user specified and returned)

Subroutine FIN(NERR,NER)

Subroutine FIN is called as the last operation in each processor to close the files used by the processor. If it is not called, these files might not be accessible to the next processor. When FIN is called at the completion of the processor task, NERR and NER are equal to 0. Subroutine FIN is also called to abort the run after a fatal error is discovered. In this case NERR and NER are printed out in format A4,I10 as a diagnostic for the error. Subroutine FIN causes printing of the time on the clock measuring CPU time for the job as well as the cumulative count of times data were written onto disk or read into central memory during execution of a processor. FIN calls TCLOCK and DATIM for this purpose (see appendix C), although they are not shown in figure 4 since they are not directly involved with data handling.

APPENDIX B

SPAR ROUTINES TO PERFORM DATA HANDLING SUPPORT FUNCTIONS

Several routines are used to perform support functions for the five routines that are needed in user programs and were described in appendix A. These support routines are normally not called directly by a user program. However, a detailed description of these routines is included in this appendix to provide a more complete definition of the operation of the SPAR data handling utilities shown in figure 4. Such information on the more basic routines is needed when transferring this system to a new host computer operating system.

Function MATCH(NU)

Function MATCH returns the line number (currently between 1 and 32) from a segment of the table of contents which is resident in central memory for library NU. The line number corresponds to the data set name which must be passed to the function in the last four words of COMMON/TOCLIN/LINE(12). Function MATCH searches successive lines of successive TOC segments, starting with the first segment in library NU, until a match of data set names is found. If the specified data set name is not found, MATCH is set to 0 on return. Function MATCH calls RDIND to read successive segments of the table of contents.

Function NTOC(NU)

To conserve central memory, bookkeeping information for portions of the TOC for only two libraries can be resident in central memory at any one time. The NTOC routine performs the function of exchanging this bookkeeping information between disk and central memory as needed by DAL, LTOC, TOCO, and FIN. This routine returns either the value 1 or 2, which corresponds to the user-specified library number NU and reads the required information into central memory if it is not already there. The information which is read into central memory includes the master directory MASTER(64,2) and a segment of the table of contents IND(12,32,2), which are both contained in COMMON/CLIB/...

Subroutine RDIND(N,IBLK)

Subroutine RDIND performs the function of reading the data in the segment of a table of contents denoted IBLK from the library corresponding to N (either 1 or 2), where N is equal to NTOC(NU). These data are read into central memory starting at IND(1,1,N). This subroutine also updates and writes the master directory MASTER(64,N) from central memory to disk each time a new data set is added to the library.

APPENDIX B

Subroutine WRTIND(N)

Subroutine WRTIND writes the information contained in the segment of the table of contents which is in central memory and whose library corresponds to N onto the library file on disk.

Subroutine STATIO(NU,ITYPE,IASG)

Subroutine STATIO is used to cause the file with the library number NU to be opened or closed.

ITYPE — Code for opening or closing the file:

≤0	close the file and IASG will be returned to zero
1	open a direct-access file
2	open a sequential file

IASG — Error code for file opening:

0	an end of file was encountered while attempting to do a check or to read the file environment of a direct-access file after opening it
1	a direct-access file is successfully opened

Subroutine WR(NU,KA,LE,NS,IOP,ISTAT,NWTX)

Subroutine WR directly performs transfer of data between central memory and disk in the UNIVAC version. For CDC and Prime versions, this subroutine serves as an intermediary between RIO and other machine-dependent routines that perform the transfers. All data transfers are performed using an integer number of disk sectors. The number of disk sectors required for transferring LE words is calculated in WR. If LE is not an integer multiple of the sector length, the last partially filled sector is temporarily stored in COMMON/B64/L64(64) and the filled portion of this sector is then added onto the array KA. For the CDC version, the disk address NS at which the transfer is to occur is passed to the COMPASS subroutines in IDX(2), which is contained in COMMON/PAK.

Function NSECTS(L)

Function NSECTS returns the number of disk sectors required to contain L words. The number of words per sector is given in LSECT in COMMON/CINDEX/ (LSECT = 28, 55, and 64 for UNIVAC, Prime, and CDC versions, respectively).

Function LADJ(L)

Function LADJ returns the total number of words available on the disk sectors which are required to contain L words. The relationship between the functions LADJ and NSECTS is LADJ(L) = LSECT*NSECTS(L).

APPENDIX B

Other Routines

A detailed description of the formal parameters in the call sequences of the machine-dependent COMPASS subroutines for the CDC version are not included herein. The general function they perform is as follows:

XOPEN opens sequential and direct-access files

XREWIND rewinds files

XEVICT closes files

READX reads data from disk to central memory

WRITEX writes data from central memory to disk

WRTINX is a write-in-place subroutine for replacing existing records on a file

The routines are used for both sequential and direct-access files. The read and write operations do not require a buffer area in the user's program.

APPENDIX C

ROUTINES TO INITIALIZE SPAR PROCESSORS AND READ USER INPUT

In addition to the data handling routines there are SPAR routines which are used to perform the initialization of the processors and to read user-prepared data from the input file. The relationships among these routines are shown schematically in figure 5. Subroutine RSET is called at the beginning of each processor. Subsequently, RSET calls KOREFL (on CDC) or KSIZE (on UNIVAC) or uses LOC directly (on Prime) to calculate the amount of blank COMMON storage locations available. On the UNIVAC system, RSET calls KEXP to expand the available central memory as requested by the user. Routines TCLOCK, DATIM, and IA6 are also called by RSET to supply the date and time entries for the table of contents of all data sets produced by that processor.

Subroutine READER is used throughout the processors to read all data from the input file. Subroutine READ actually causes a record to be read and function KALPH is used in assembling alphanumeric words as the record is being interpreted by READER.

Subroutine RSET(IL,M,IEA)

Subroutine RSET is used to read RESET statements which appear on the input file following the @XQT command for each processor. Subroutine RSET also causes the input record following the last RESET statement to be read. Parameters used in this reading process are passed to the subroutine through the array IL and integer M. These values are defined in DATA statements in the user program.

M — Number of reset parameters to be used

IL(1,I) — Contains the four-character alphanumeric name of the Ith reset parameter ($1 \leq I \leq M$)

IL(2,I) — Contains the type code for the parameter:

0	integer
-1	real
4	alphanumeric

IL(3,I) — Contains the actual value for the parameter; the default value which is defined in the user program is replaced by the value, if any, which is read from the RESET statement which is input

IEA — Error return indicator:

0	program execution will terminate if an error is detected on a RESET statement
$\neq 0$	an error on a RESET statement is ignored

APPENDIX C

Subroutine KOREFL(A,KORE,KFL)
(This subroutine used in CDC version only)

This COMPASS subroutine calculates the amount of central memory KORE between the address of A and the end of the user-specified field length KFL. KORE is calculated by the equation

KORE=KFL-(Address of A)- 1

This subroutine is called by RSET in SPAR to determine the amount of blank COMMON which is available to be allocated for working storage.

Subroutine TCLOCK(I,CP,DCP)

Subroutine TCLOCK returns the current time from the CPU clock for a job. DCP is the amount of CPU time that has been used since the last call to this subroutine. If I is equal to 1, DATIM is called to calculate the current date and time which is then stored in COMMON/CIDT/IDATE,ITIME. If I is equal to 3, DCP and CP are printed.

Subroutine DATIM(IDATE,ITIME)

This subroutine returns the current date and wall clock time as alphanumeric words in the form 77/03/17. and 11.12.05.

Function IA6(IA10)
(This function used in CDC version only)

Function IA6 converts a 10-character word like 0123456789 to a 6-character word in the form 124578. The function is used in SPAR to change a date from the alphanumeric form 77/03/17. to the integer 770317 and time from 11.12.05. to 111205. These integer forms of date and time are used as entries in the tables of contents for data libraries.

Subroutine READER

Subroutine READER interprets the information on an input record having a free-field format which follows the rules given in the SPAR Reference Manual (ref. 5). The decoded information is returned in COMMON/INREC/. (See appendix D for a description of the contents of this COMMON block.)

Subroutine READ(IA,IEOF)

Subroutine READ reads an 80-character record from the input file into an 80-word array IA. The parameter IEOF is 0 for a normal read and 1 if an end-of-file is hit when attempting to read.

APPENDIX C

Function KALPH(IN,N)

Function KALPH assembles the N characters contained as separate words in the array IN into a single left-adjusted word. The integer N must be equal to or less than 4.

APPENDIX D

CONTENTS OF LABELED COMMON BLOCKS

This appendix contains the contents of labeled COMMON blocks which are used for communication among the routines shown in figures 4 and 5. These blocks are presented in alphabetical order of the block name for each group of routines.

COMMON Blocks Used by Data Handling Routines of Figure 4

(Blocks CFIN and CIDT are used by some of these routines but are explained in the next section.) The COMMON blocks used by the routines of figure 4 are given below along with a description of their contents.

COMMON/B64/L64(64)

L64 Used for temporary storage of words from a disk sector that is partially full (see discussion of subroutine WR in appendix B)

COMMON/CLIB/NSWAP,NUNS,NUN(2),NOP(2),NBLOKS(2),NWRITE(2),INCORE(2),NMAST,
MASTER(64,2)INDSZ,NIND,LIND,IND(12,32,2)

This COMMON block contains information on the two segments of the TOC's from two different libraries currently resident in central memory (see also section "Description of SPAR Data Handling Utilities")

NSWAP Counter that is incremented in function NTOC each time a segment of the TOC of a library is exchanged between central memory and disk

NUNS Fixed value of 2, indicating that the master directory and a segment of TOC (containing 32 entries) from two libraries can be in central memory at the same time

NUN(I) Contains the library number which corresponds to the Ith (1 or 2) set of directory and TOC information currently in central memory

NOP(I) Counter which indicates the relative activity or use of the two libraries; the library with least activity is replaced when a new one is required

NBLOKS(I) Number of segments in the table of contents for the library; it is identical with MASTER (3,I)

NWRITE(I) Set to 0 when the segment of the table of contents IND(12,32,I) is written to disk, or is set to 1 when the master directory MASTER(64,I) is written to disk (where I = 1 or 2)

INCORE(I) Contains the sequence number of the segment for the table of contents of the library currently in central memory

NMAST Number of entries for each of the two libraries in MASTER; presently fixed at 64

MASTER(64,I) Master directory for the two libraries (see the section entitled "Master Directory")

INDSZ Number of entries in a line of the table of contents; presently fixed at 12

NIND Number of lines per segment of the table of contents; presently fixed at 32

LIND Total number of words in a segment of the table of contents; LIND=INDSZ*NIND, presently equal to 384

IND(12,32,I) Segment of the table of contents (see the section "Table of Contents")

COMMON/CINDEX/INDEX(7,30),LSECT

INDEX(1,I) Not defined (I refers to Ith user-specified library number)

INDEX(2,I) Current location of the disk address pointer

INDEX(3,I) Next available (not yet used) address in library I

INDEX(4,I) Counter for the number of read statements from the library

INDEX(5,I) Counter for the number of write statements to the library

INDEX(6,I) File status indicator:

-N	a file has been opened and is now closed
0	a file has not been opened
N	a file is currently open, where N is the user library number; for direct-access files, $1 \leq N \leq 26$; for a sequential file, N = 10 000

INDEX(7,I) Order of assignment of files which are currently active; the number of entries in the NASG array

LSECT Number of words per sector on the auxiliary storage device (28 words for UNIVAC, 55 words for Prime, and 64 words for CDC)

COMMON/NAMASK/MASK,LUNIT

MASK Equal to 4HMASK

LUNIT Equal to
6HSPARL for CDC version
4HSPLA for Prime version
6HSPAR for UNIVAC version

COMMON/PAK/IDX(2),NPAKS,NASG(10),IPAK(17,10)

This COMMON block is used by subroutines WR and STATIO to communicate with the COMPASS language I/O subroutines

IDX(1) Not defined

IDX(2) Used to pass the disk address at which a read or write operation is to occur

NPAKS Maximum number of active files allowed; presently set at 10

NASG(I) User-specified library number for the Ith active file that is currently open

IPAK(17,I) 17 words comprising the file environmental tables (FET) for the Ith active file

COMMON/TOCLIN/LINE(12)

LINE(I) Ith entry in a line of the TOC (see section entitled "Table of Contents")

COMMON Blocks Used by Subroutines Shown in Figure 5

The COMMON blocks used by the routines of figure 5 are given below along with a description of their contents.

COMMON/CFIN/IABORT,IOPRT

IABORT Flag which has a default value of zero but can be changed in a RESET record, for any processor, to a value of unity and the processor will not make an error abort if it encounters a serious error (e.g., if the required data sets do not exist)

IOPRT Similar to IABORT, except a value of unity causes the processor to print extra I/O information

COMMON/CFMT/KALT(10)

KALT Contains user-specified global control parameters for the processors; currently, only three parameters are used

KALT(1) Contains the value specified on a FORMAT input record; this parameter is used in different processors for different purposes (see the TAB and SA processors in SPAR)

KALT(2) Contains the value specified on an ONLINE input record; the value is 0 for minimum printout, 1 for normal printout, or 2 for maximum printout

KALT(3) Contains the value specified on an IOUT input record

COMMON/CIDT/IDATE,ITIME

IDATE Contains an integer date in the form 770317

ITIME Contains an integer time in the form 111205 (see the descriptions of TCLOCK, DATIM, and IA6 in appendix C)

COMMON/INREC/IDATA(40),KIND(40),NAME,NDW,NA41,NA42,NRPR,ICHAR(81)

IDATA Contains up to 40 words which could be put on an 80-column input card, since words must be separated by blanks, commas, etc. in the SPAR free-field input format

KIND Contains the type code for each of the words in IDATA

NAME Contains IDATA(1) if it is an alphanumeric word; otherwise it is zero

NDW Number of words used in the IDATA array

NA41 Word number in IDATA where four-character words in the comment field start

NA42 Word number where the comment field ends

NRPR Contains the number of input records on the input card which was read

ICHAR Contains the 80 characters from an input card with a $ in ICHAR(81)

APPENDIX E

LISTING OF SAMPLE PROCESSOR

The sample processor contained in this appendix illustrates the use of the SPAR data handling routines. The five data handling routines (RIO, DAL, LTOC, TOCO, and FIN) described in appendix A are included in this processor as well as the input routines (RSET and READER), which are described in appendix C.

The function of the processor is to create a library and subsequently read and print a data set containing a vector of real numbers and a data set containing a matrix of integers. Comments are included in the listing, which follows, to give a detailed description of the steps performed by the processor.

```
      PROGRAM TEST
      INTEGER VG01,VG02
      COMMON/INREC/ IDATA(40),KIND(40),NAME,NDW,NA41,NA42,NRPR,ICHAR(81)
      COMMON/TOCLIN/ LINE(12)
      COMMON KORE,KEVEN,A(1)
      DIMENSION KA(1)
      DIMENSION CDATA(40)
      DIMENSION NA4(4)
      DIMENSION IV(3,4),RV(3,4)
      EQUIVALENCE (A(1),KA(1))
      EQUIVALENCE (IDATA(1),CDATA(1))
      EQUIVALENCE (IV(1,1),RV(1,1))
      DATA NL/4/
      DATA IV
     $/4HNLIB, 0, 1,      4HNRV , 0, 0
     $,4HNRM , 0, 0,      4HNCM , 0, 0
     $/
C
C     TO READ RESET VALUES AND FIRST DATA CARD
C
      CALL RSET(IV,NL,0)
      NU = IV(3,1)
      NROWV = IV(3,2)
      NROWM = IV(3,3)
      NCOLM = IV(3,4)
C
C     TO SET UP POINTERS IN BLANK COMMON
C
      VG01 = 1
      MG01 = VG01+NROWV
      VG02 = MG01+NROWM*NCOLM
      MG02 = VG02+NROWV
C
C     TO PUT VECTOR IN BLANK COMMON STARTING AT VG01
C
      DO 10 I=1,NROWV
      J = VG01+I-1
   10 A(J) = CDATA(I)
```

```
C
C       TO PUT MATRIX IN BLANK COMMON STARTING AT MGO1
C
        DO 20 I=1,NCOLM
        K = MGO1+NROWM*(I-1)
        CALL READER
        DO 20 J=1,NROWM
        L = K+J-1
   20   KA(L) = IDATA(J)
C
C       TO WRITE VECTOR ON DISK
C
        CALL DAL(NU,1,A(VGO1),0,1,KADR,IERR,NROWV,1,NROWV,-1,
       $4HTEST,4HVEC ,1,1)
C
C       TO WRITE MATRIX ON DISK IN BLOCKED FORM
C
        NWDS = NROWM*NCOLM
        CALL DAL(NU,0,A,0,1,KADR,IERR,NWDS,1,NROWM,0,
       $4HTEST,4HMAT ,1,1)
        DO 30 I=1,NCOLM
        J = MGO1+NROWM*(I-1)
   30   CALL RIO(NU,10,2,KADR,KA(J),NROWM)
C
C       TO READ VECTOR INTO BLANK COMMON STARTING AT VGO2
C
        CALL DAL(NU,11,A(VGO2),0,IEA,KADR,IERR,NWDS,NE,LB,ITYPE,
       $4HTEST,4HVEC ,1,1)
        IGO = VGO2
        IEND = IGO+NROWV-1
        WRITE(6,40) (A(I),I=IGO,IEND)
   40   FORMAT(5X,F10.2)
C
C       TO USE BOTH LTOC AND TOCO TO GET INFORMATION TO READ MATRIX
C
        KADR = LTOC(NU,1,4HTEST,4HMAT ,1,1)
        NA4(1) = 4HTEST
        NA4(2) = 4HMAT
        NA4(3) = 1
        NA4(4) = 1
        NLINE = 0
        CALL TOCO(NU,NA4,1,NLINE)
        NWDS = LINE(5)
        NE = LINE(6)
        LB = LINE(7)
        NBLKS = NWDS/LB
        IF(NBLKS*LB.NE.NWDS) NBLKS = NBLKS+1
C
C       TO READ MATRIX INTO BLANK COMMON STARTING AT MGO2
C
        DO 50 I=1,NBLKS
        J = MGO2+LB*(I-1)
   50   CALL RIO(NU,20,2,KADR,KA(J),LB)
        IGO = MGO2
        IEND = IGO+NROWM*NCOLM-1
        WRITE(6,60) (KA(I),I=IGO,IEND)
   60   FORMAT(/5X,5I7)
```

APPENDIX F

LISTING FROM INTERACTIVE EXECUTION OF SAMPLE PROCESSOR

The listing contained in this appendix is from an interactive execution of the sample processor, so that it includes both input to and output from the processor. The information following the operating system prompt (the "?" character) is input from the user. All other lines are output by the processor or related SPAR subroutines. Comments are included in the listing to describe the input that follows each comment. These comments all begin with the $ character.

Execution of the sample processor is initiated by the executive control command, [XQT TEST. (Note the CDC and Prime versions use [XQT and the UNIVAC version uses @XQT.) RESET values which control execution of the processor are then specified. The four RESET values which can be input to the sample processor are shown in the data statement containing the IV array. (See listing of the processor in appendix E.)

These RESET values are defined as follows:

Name	Default value	Meaning
NLIB	1	Library number on which data sets will be created
NRV	0	Number of rows in the vector to be input
NRM	0	Number of rows in the matrix to be input
NCM	0	Number of columns in the matrix to be input

In the listing for the interactive execution of the sample processor, the default value of NLIB = 1 is used and remaining reset values are specified by the user, as shown in the listing. The values for the vector and the matrix are then input by the user and, after being stored on and retrieved from disk, are printed by the processor.

Next an executive control command is used to execute the SPAR data complex utility (DCU) processor. User commands are input to this processor, which prints the table of contents of library 1 containing the two created data sets and also the contents of these data sets, as shown in the listing. Finally the executive control command, [XQT EXIT, is used to terminate the run. Output from execution of the sample processor is as follows:

```
? $
? $  TO EXECUTE THE TEST PROCESSOR
? $
? (XQT TEST
0*EXECUTE TEST
? $
? $  TO INPUT RESET VALUES
? $
? RESET NRV=6,NRM=5,NCM=3
 NRV =        6
 NRM =        5
 NCM =        3
? $
? $  TO INPUT VECTOR
? $
?   0.  .1  .2  .3  .4  .5
 DATA SPACE=     9664
? $
? $  TO INPUT MATRIX = BY COLUMNS
? $
?   11  21  31  41  51
?   12  22  32  42  52
?   13  23  33  43  53
           0.00
            .10
            .20
            .30
            .40
            .50

          11      21      31      41      51

          12      22      32      42      52

          13      23      33      43      53
0EXIT    1.428          8          4
```

```
? $
? $  TO CALL SPAR UTILITY PROCESSOR = DCU
? $
? IXQT DCU
0*EXECUTE DCU
? $
? $  TO PRINT TABLE OF CONTENTS
? $
? TOC 1
 DATA SPACE=     8192
0
1TABLE OF CONTENTS, LIBRARY  1
0                                E                                  T   DATA SET NAME
 SEQ       RH    DATE    TIME   R    WORDS     NJ NI*NJ   Y    N1    N2    N3    N4
   1        7 780523 092148  0        6      1      6  -1 TEST VEC      1     1
   2        8 780523 092148  0       15      1      5   0 TEST MAT      1     1
? $
? $  TO PRINT CONTENTS OF DATA SETS
? $
? PRINT 1 TEST VEC
0
 PRIN LIB/SEQ= 1/    1    TEST    VEC           1        1
1BLOCK    1  TEST    VEC           1        1
0           I=     1      I=     2      I=     3      I=     4      I=     5      I=     6
 J=     1  0.           .10000E+00  .20000E+00  .30000E+00  .40000E+00  .50000E+
0ABOVE PRODUCED FROM LIB  1
 TOC=          7  780523  092148        0        6        1        6       -1
 TEST    VEC             1        1
? PRINT 1 TEST MAT
0
 PRIN LIB/SEQ= 1/    2    TEST    MAT           1        1
1BLOCK    1  TEST    MAT           1        1
0           I=     1      I=     2      I=     3      I=     4      I=     5      I=
 J=     1         11            21            31            41            51
1BLOCK    2  TEST    MAT           1        1
0           I=     1      I=     2      I=     3      I=     4      I=     5      I=
 J=     1         12            22            32            42            52
1BLOCK    3  TEST    MAT           1        1
0           I=     1      I=     2      I=     3      I=     4      I=     5      I=
 J=     1         13            23            33            43            53
0ABOVE PRODUCED FROM LIB  1
 TOC=          8  780523  092148        0       15        1        5        0
 TEST    MAT             1        1
? IXQT EXIT
0EXIT    1.518          9          8
```

APPENDIX G

DYNAMIC STORAGE ALLOCATION

Dynamic storage allocation is a programing technique to stack all problem-size dependent arrays into the blank COMMON area of central memory in an efficient manner. The amount of blank COMMON that is allocated for an array is the minimum needed to accommodate the array size for a given problem. This technique eliminates most dimension statements with fixed-size arrays.

The SPAR system uses dynamic storage allocation, since the size of most of the arrays that are used depends on the size (number of joints and elements) of the finite-element model being analyzed. This technique is not unique to SPAR; that is, many other programs or systems have used it. However, since this technique is very desirable for use with the SPAR data handling routines, a description of the basic principles of dynamic allocation is included herein for completeness.

Use of the technique is illustrated by the following example program. This program performs the function of reading joint coordinates and joint deflections, adding them together to give the deformed shape, which is printed. The program provides dynamic storage allocation for the arrays containing the joint coordinates, deflections, and deformed shape. A listing of the program is given, followed by an explanation of the statements which are important in the allocation process.

```
      PROGRAM EXAMPLE(TAPE5=INPUT,TAPE6=OUTPUT)
      INTEGER OLD,DEFL
      COMMON A(1)
      READ(5,1) NJTS
    1 FORMAT(I5)
      OLD = 1
      DEFL = OLD+NJTS*3
      NEW = DEFL+NJTS*6
      NEND = NEW+NJTS*3
      CALL SUM(A(OLD),A(DEFL),A(NEW),NJTS)
      IGO = NEW
      IEND = NEND-1
      WRITE(6,2) (A(I),I=IGO,IEND)
    2 FORMAT(5X,6E20.8)
      END


      SUBROUTINE SUM(XYZ,DEFL,TOTL,NJTS)
      DIMENSION XYZ(NJTS,3),DEFL(NJTS,6),TOTL(NJTS,3)
      READ(5,1) ((XYZ(I,J),J=1,3),I=1,NJTS)
    1 FORMAT(3F10.5)
      READ(5,2) ((DEFL(I,J),J=1,6),I=1,NJTS)
    2 FORMAT(6F10.5)
      DO 10 I=1,NJTS
      DO 10 J=1,3
   10 TOTL(I,J) = XYZ(I,J)+DEFL(I,J)
      RETURN
      END
```

The blank COMMON area is designated as array A. On CDC systems, A only needs to be dimensioned at A(1), since it actually extends to the end of the field length for the job. For UNIVAC programs, A is typically dimensioned to accommodate a small or medium-size problem. Core expanding routines (such as KEXP) may be used for large problems. Alternatively, the main program may be recompiled with a larger value in the COMMON statements when a large problem is solved. The problem-size-dependent parameter (NJTS), which indicates the number of joints to be considered, is read from the input file. In SPAR, such parameters usually come from the table of contents. The pointers OLD, DEFL, and NEW are calculated as the starting addresses in array A for the array containing joint locations, deflections, and deformed shape, respectively. The total amount of blank COMMON required is given by NEND. These array starting addresses are passed to the subroutine SUM, which actually performs the addition. In subroutine SUM, these arrays have names which reflect the information they contain. The dimension statement is required for names which are arrays in the parameter list of SUM.

Conversion of an existing program to provide dynamic storage allocation generally requires a subroutine to set up the starting addresses of arrays and to call other subroutines using these addresses in the calling sequences. The calling sequences often get quite long if many arrays with different names are used. For subroutines performing the calculations, the proper respective formal parameters in the subroutine statement must be provided, and dimension statements indicating that these parameters are array names must be added.

REFERENCES

1. Butler, Thomas G.; and Michel, Douglas: NASTRAN - A Summary of the Functions and Capabilities of the NASA Structural Analysis Computer System. NASA SP-260, 1971.

2. Miller, Ralph E., Jr.: Structures Technology and the Impact of Computers. Integrated Design and Analysis of Aerospace Structures, R. F. Hartung, ed., American Soc. Mech. Eng., c.1975, pp. 57-70.

3. Sobieszczanski, Jaroslaw: Building a Computer-Aided Design Capability Using a Standard Time Share Operating System. Integrated Design and Analysis of Aerospace Structures, R. F. Hartung, ed., American Soc. Mech. Eng., c.1975, pp. 93-112.

4. Giles, Gary L.: Computer-Aided Methods for Analysis and Synthesis of Supersonic Cruise Aircraft Structures. Proceedings of the SCAR Conference - Part 2, NASA CP-001, [1977], pp. 637-657.

5. Whetstone, W. D.: SPAR Structural Analysis System Reference Manual - System Level II. Volume I - Program Execution. NASA CR-145096-1, 1977.

6. Storaasli, Olaf O.; and Foster, Edwin P.: Cost-Effective Use of Minicomputers To Solve Structural Problems. AIAA Paper No. 78-484, Apr. 1978.

7. Whetstone, W. D.: Computer Analysis of Large Linear Frames. J. Struct. Div., American Soc. Civil Eng., vol. 95, no. ST11, Nov. 1969, pp. 2401-2417.

8. Whetstone, William D.; and Jones, Charles D.: Vibrational Characteristics of Linear Space Frames. J. Struct. Div., American Soc. Civil Eng., vol. 95, no. ST10, Oct. 1969, pp. 2077-2091.

TABLE 1.- SPAR PROCESSORS

Processor name	Function
TAB	Creates data sets containing tables of joint locations, section properties, material constants, etc.
ELD	Defines the finite elements making up the model
E	Generates sets of information for each element, including connected joint numbers, geometrical data, material and section property data
EKS	Adds the stiffness and stress matrices for each element to the set of information produced by the E processor
TOPO	Analyzes element interconnection topology and creates data sets used to assemble and factor the system mass and stiffness matrices
K	Assembles the unconstrained system stiffness matrix in a sparse format
M	Assembles the unconstrained system mass matrix in a sparse format
KG	Assembles the unconstrained system initial-stress (geometric) stiffness matrix in a sparse format
INV	Factors the assembled system matrices
EQNF	Computes equivalent joint loading associated with thermal, dislocational, and pressure loading
SSOL	Computes displacements and reactions due to loading applied at the joints

TABLE 1.- Concluded

Processor name	Function
GSF	Generates element stresses and internal loads
PSF	Prints the information generated by the GSF processor
EIG	Solves linear vibration and bifurcation buckling eigenproblems
DR	Performs a dynamic response analysis
SYN	Produces mass and stiffness matrices for systems comprised of interconnected substructures
STRP	Computes eigenvalues and eigenvectors of substructured systems
AUS	Performs an array of matrix arithmetic functions and is used in construction, editing, and modification of data sets
DCU	Performs an array of data management functions including display of table of contents, data transfer between libraries, changing data set names, printing data sets, and transferring data between libraries and sequential files
VPRT	Performs editing and printing of data sets which are in the form of vectors on the data libraries
PLTA	Produces data sets containing plot specifications
PLTB	Generates the graphical displays which are specified by the PLTA processor

TABLE 2.- TABLE OF CONTENTS FOR SPAR DATA LIBRARY

SEQ	RR	DATE	TIME	ER	WORDS	NJ	NI*NJ	TY	DATA SET NAME N1	N2	N3	N4
1	7	761022	081237	0	18	1	18	0	JDF1	BTAB	1	8
2	-8	761022	081237	0	250	250	250	0	JREF	BTAB	2	6
3	-12	761022	081237	0	12	1	12	1	ALTR	BTAB	2	4
4	13	761022	081237	0	30	2	30	4	TEXT	BTAB	2	1
5	14	761022	081237	0	10	1	10	1	MATC	BTAB	2	2
6	15	761022	081237	0	48	4	48	1	ALTR	BTAB	2	4
7	16	761022	081237	0	750	250	750	1	JLOC	BTAB	2	5
8	28	761022	081237	0	250	250	250	0	JREF	BTAB	2	6
9	32	761022	081237	0	25	1	25	1	SA	BTAB	2	13
10	33	761022	081237	0	250	250	250	0	CON		1	0
11	37	761022	081237	0	2250	250	2250	1	QJJT	BTAB	2	19
12	73	761022	081241	0	3136	196	896	0	DEF	E43	11	4
13	129	761022	081241	0	2	1	2	0	GD	E43	11	4
14	130	761022	081241	0	15	1	15	0	GTIT	E43	11	4
15	131	761022	081241	0	12	12	12	0	NELZ	BTAB	1	11
16	132	761022	081241	0	5	1	5	0	KE		0	0
17	133	761022	081241	0	7	1	7	0	NS		0	0
18	134	761022	081241	0	1	1	1	3	ELTS	NAME	0	0
19	135	761022	081241	0	1	1	1	0	ELTS	LTYP	0	0
20	136	761022	081241	0	1	1	1	0	ELTS	NNOD	0	0
21	137	761022	081241	0	1	1	1	0	ELTS	ISCT	0	0
22	138	761022	081241	0	1	1	1	0	ELTS	NELS	0	0
23	139	761022	081241	0	1	1	1	0	ELTS	LE3	0	0
24	140	761022	081244	0	1500	250	1500	-1	APPL	FORC	1	1
25	164	761022	081244	0	1500	250	1500	-1	APPL	FORC	2	1
26	188	761022	081244	0	1500	250	1500	-1	UNIT	VEC	1	1
27	212	761022	081244	0	1500	250	1500	-1	APPL	MOTI	5	1
28	236	761022	081244	0	1	1	1	-1	TOT	FORC	5	1
29	237	761022	081244	0	1	1	1	-1	CONV	AUS	1	1
30	238	761022	081244	0	1500	250	1500	-1	DISP	INC	1	1
31	262	761022	081245	0	6272	250	896	0	KMAP		1087	17
32	360	761022	081245	0	8960	250	1792	0	AMAP		1675	28
33	506	761022	081254	0	62720	196	320	4	E43	EFIL	11	4
34	1486	761022	081248	0	20	20	20	0	DIR	E43	11	4
35	1487	761022	081248	0	1500	250	1500	-1	DEM	DIAG	0	0
36	1511	761022	081302	0	42560	250	2240	1	K	SPAR	36	0
37	2176	761022	081323	0	50176	250	3584	1	INV	K	1	0
38	2960	761022	081333	0	1500	250	1500	-1	STAT	DISP	100	1
39	2984	761022	081333	0	1500	250	1500	-1	STAT	REAC	2	1
40	3008	761022	081345	0	10780	196	5555	-1	STRS	E43	100	1

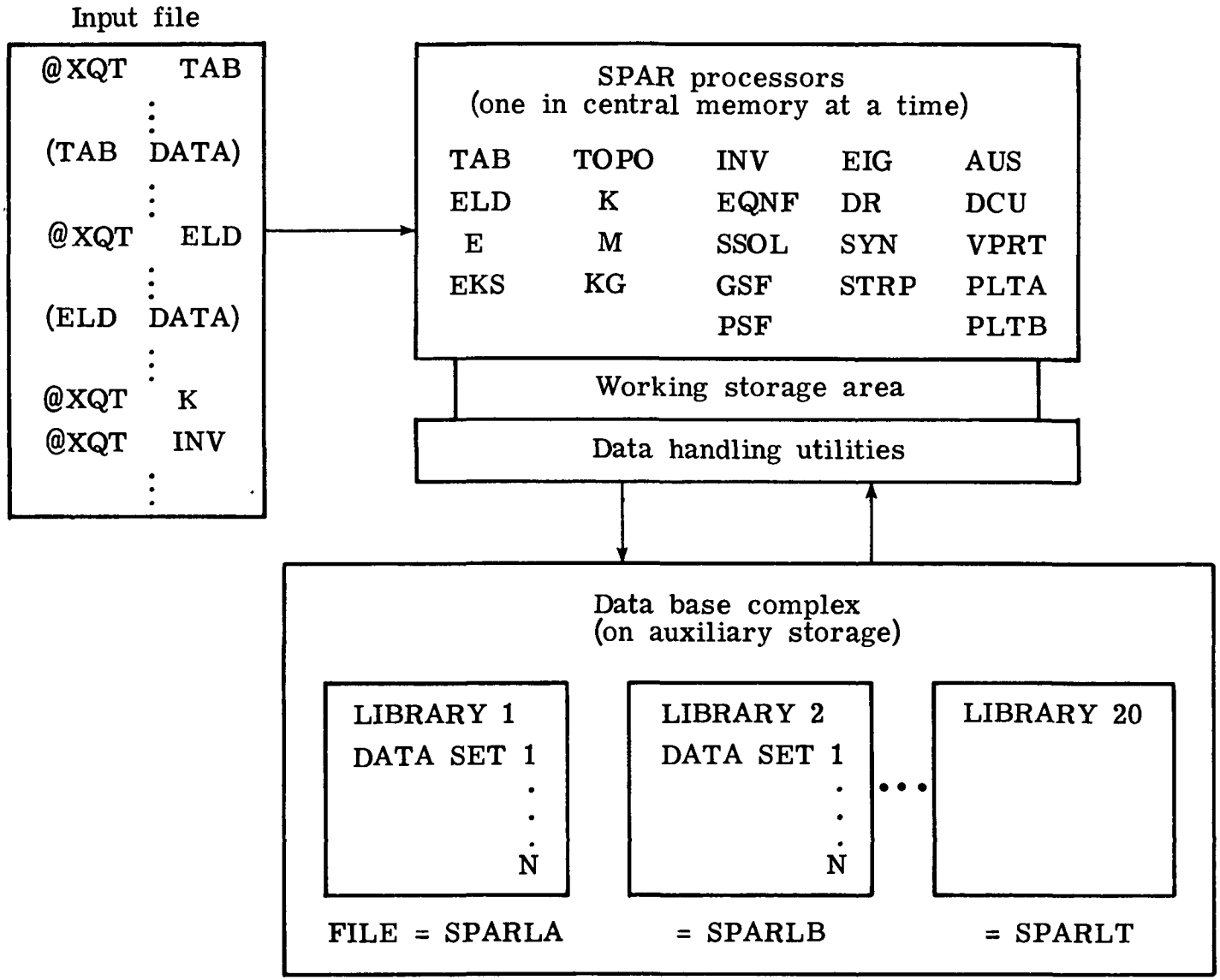


Figure 1.- Organization of SPAR analysis system.

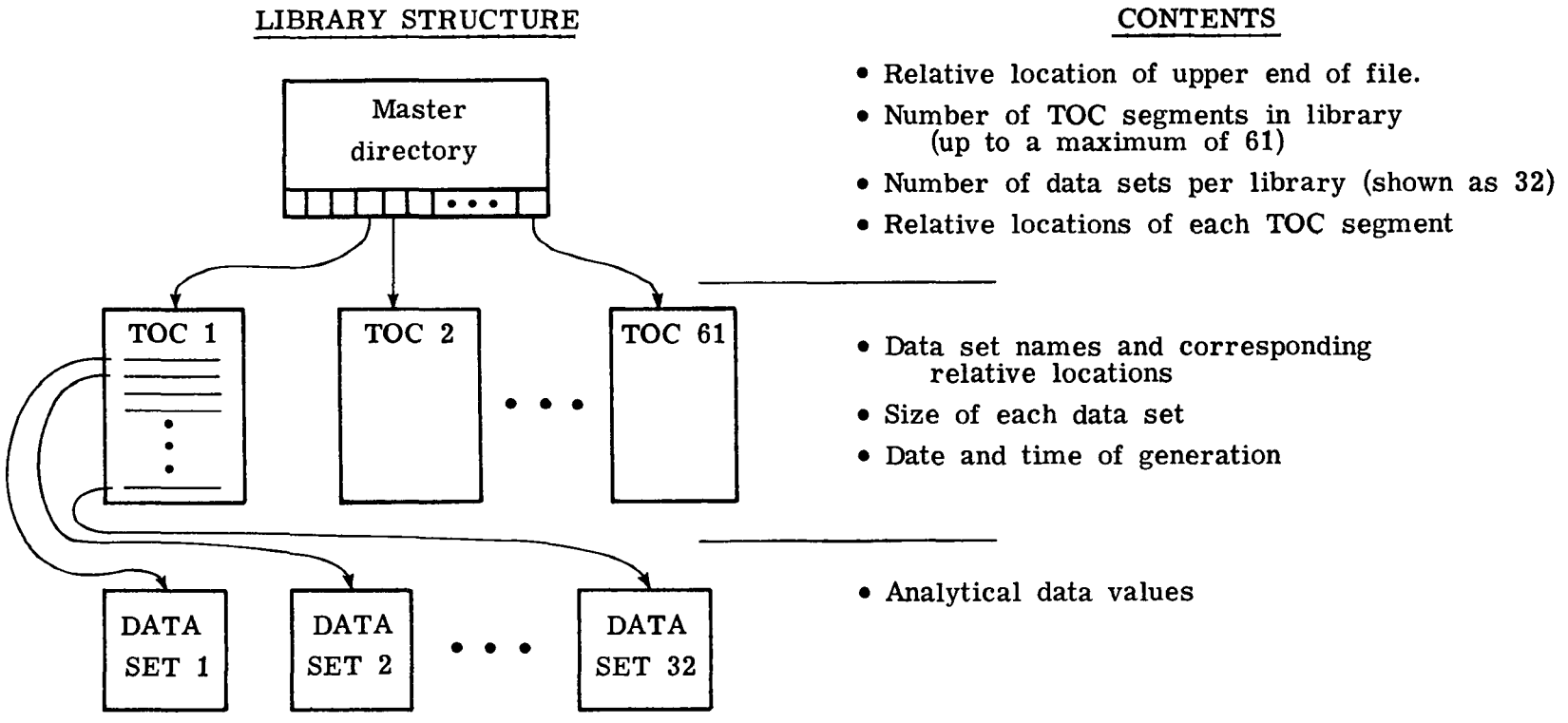


Figure 2.- Structure and contents of SPAR library.

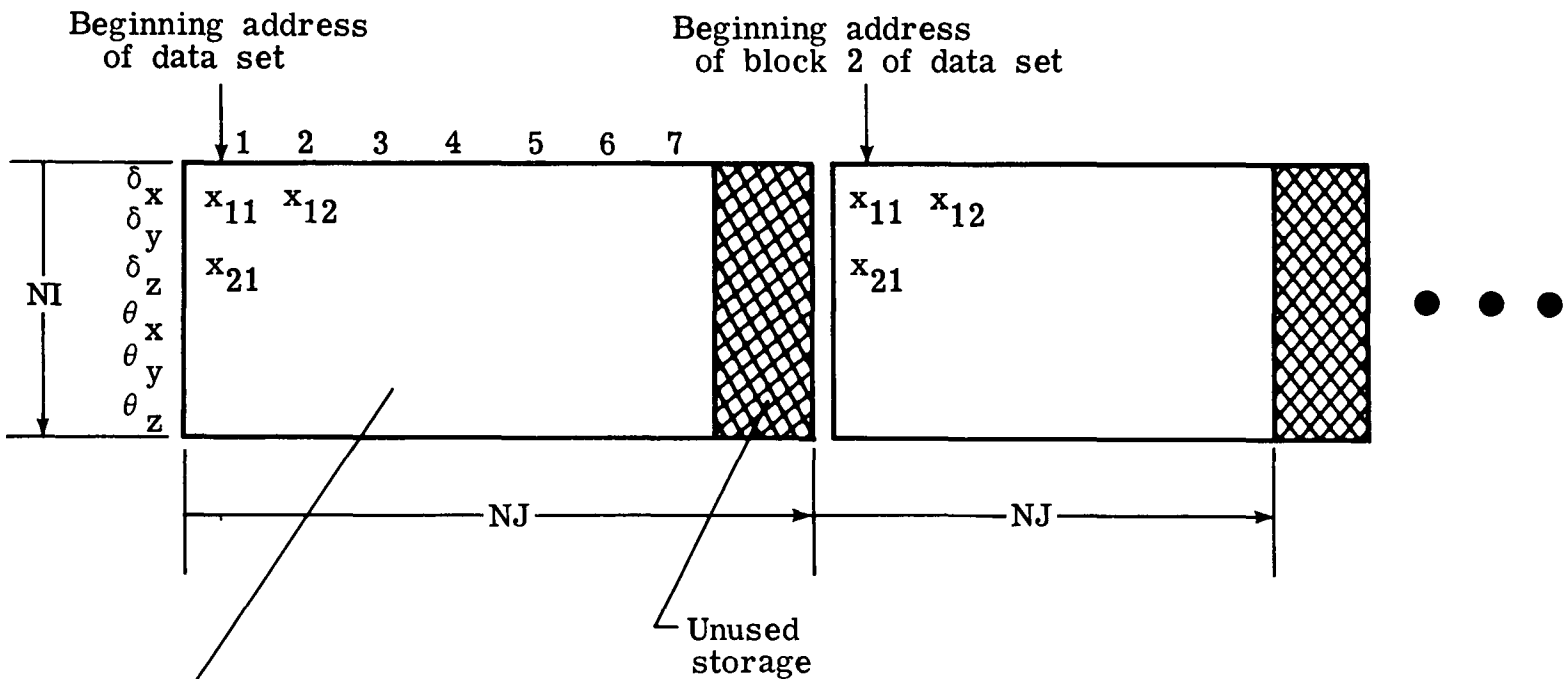


Figure 3.- Organization of a SPAR data set.

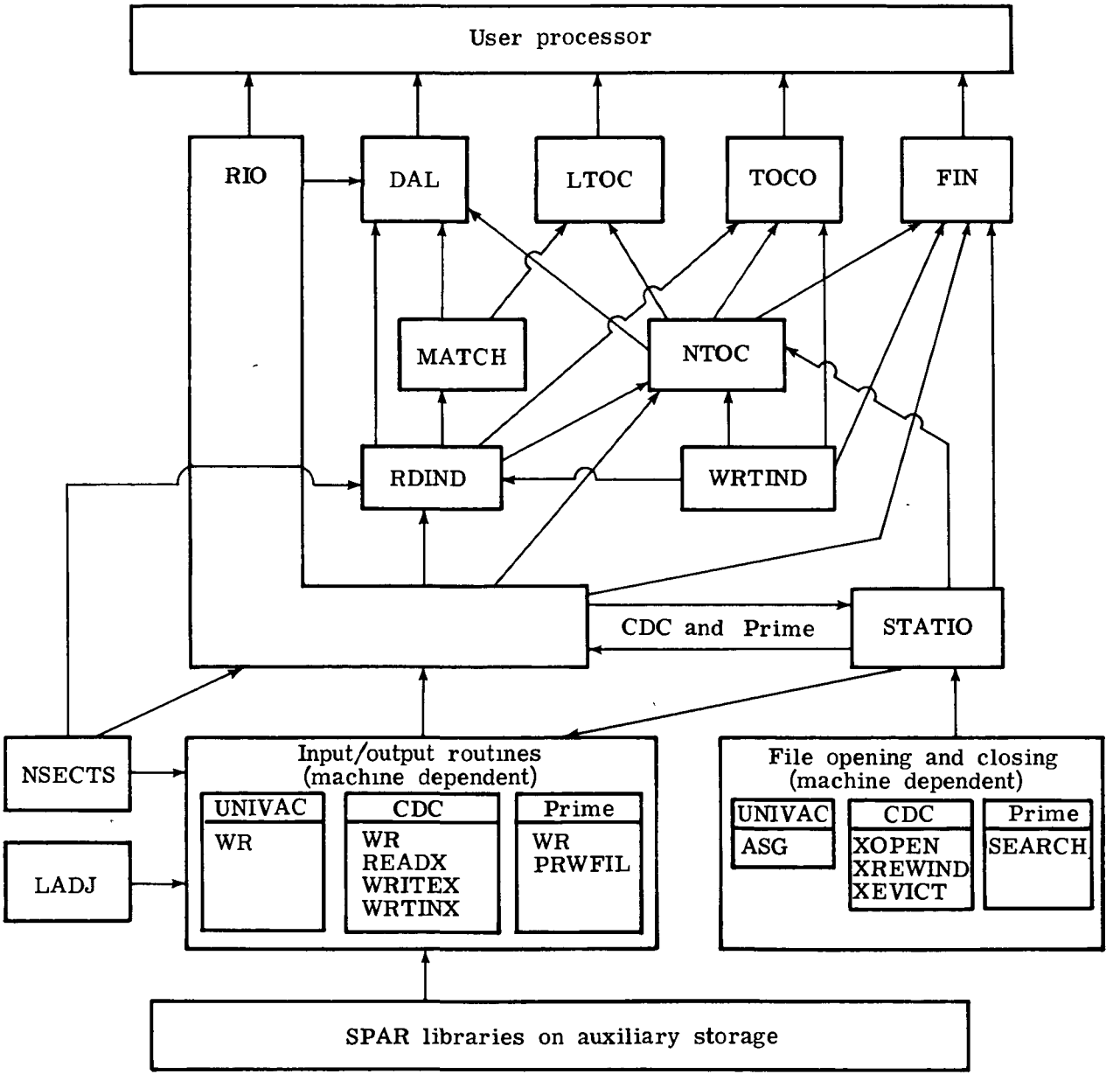


Figure 4.- Relationships of SPAR data handling routines.

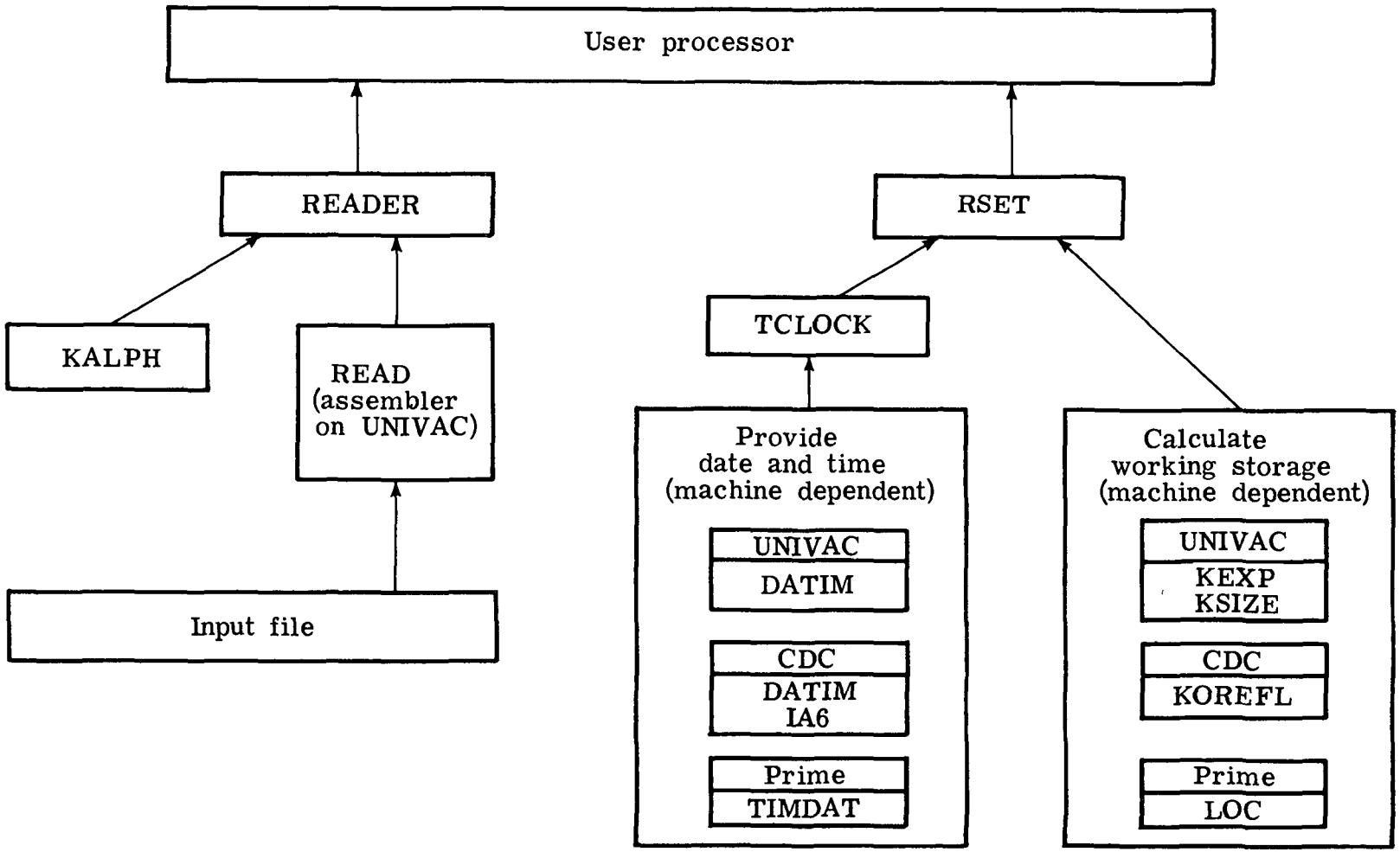


Figure 5.- Routines used to initialize SPAR processors and to read information from input file.

1 Report No	2 Government Accession No	3 Recipient's Catalog No
NASA TM-78701		

4 Title and Subtitle	5 Report Date
SPAR DATA HANDLING UTILITIES	September 1978
	6 Performing Organization Code

7 Author(s)	8 Performing Organization Report No
Gary L. Giles and Raphael T. Haftka	L-12106
	10 Work Unit No 743-01-03-05
9 Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665	11 Contract or Grant No
	13 Type of Report and Period Covered Technical Memorandum
12 Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14 Sponsoring Agency Code

15 Supplementary Notes

Raphael T. Haftka: Illinois Institute of Technology, Chicago, Illinois.

16 Abstract

The SPAR computer software system is a collection of processors that perform particular steps in the finite-element structural analysis procedure. The data generated by each processor are stored on a data base complex residing on an auxiliary storage device, and these data are then used by subsequent processors. The computer software associated with the data base complex provides a general capability and can be used for management of data in programs or systems other than SPAR.

This report documents the SPAR data handling utilities, which are routines used to transfer data between the processors and the data base complex. A detailed description of the data base complex organization is also presented. A discussion of how these SPAR data handling utilities are used in an application program to perform desired user functions is given with the steps necessary to convert an existing program to a SPAR processor by incorporating these utilities. Finally, a sample SPAR processor is included to illustrate the use of the data handling utilities.

17 Key Words (Suggested by Author(s))	18 Distribution Statement
Data handling Data base Data management Data transfer	Unclassified - Unlimited Subject Category 61

19 Security Classif (of this report)	20 Security Classif (of this page)	21 No of Pages	22 Price*
Unclassified	Unclassified	46	$4.50

* For sale by the National Technical Information Service, Springfield Virginia 22161

NASA-Langley, 1978

R. D. Welling 345/33/35/48996

National Aeronautics and
Space Administration

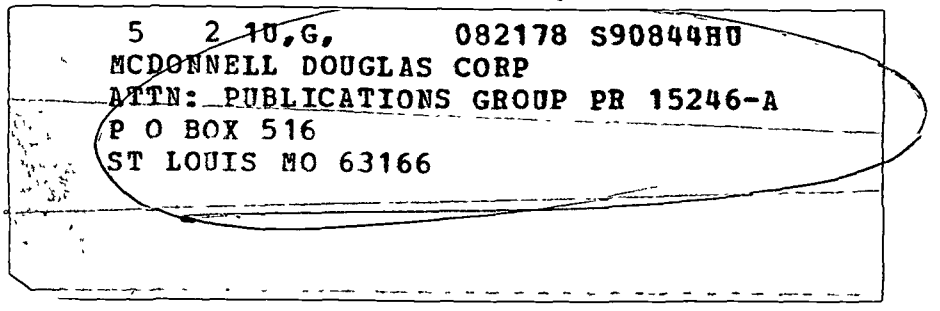
Washington, D.C.
20546

Official Business
Penalty for Private Use, $300

THIRD-CLASS BULK RATE

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

U.S.MAIL



NASA

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return

F. D. Utterback K253/73

Mark Fisher 334/1 3/246