THESIS

IMPLEMENTATION OF THE GRAPHICS-ORIENTED INTERACTIVE FINITE ELEMENT TIME-SHARING SYSTEM (GIFTS) ON THE PDP-11.

by

John Trevor Sheldon

September-1980

Thesis Advisor: Gilles Cantin

Approved for public release; distribution unlimited
# Title
Implementation of the Graphics-oriented Interactive Finite Element Time-sharing System (GIFTS) on the PDP-11

# Authors
John Trevor Sheldon

# Performing Organization Name and Address
Naval Postgraduate School
Monterey, California 93940

# Monitoring Agency Name and Address
Naval Postgraduate School
Monterey, California 93940

# Approved for public release; distribution unlimited.

# Key Words
- GIFTS
- Graphics
- Finite Element
- Minicomputer
- Structural Analysis
- PDP-11

# Abstract
The Graphics-oriented, Interactive, Finite Element Time-sharing System (GIFTS), written by Professor J. Imai and Dr. Michael McCall of the University of Arizona, has been implemented on the PDP-11 at the Naval Postgraduate School. This powerful system of programs was installed in a manner to facilitate its modification in the future. A very brief description of the GIFTS system, including the Unified Data Base, as well as the PDP-11 and RSX-11M.
operating system, are provided. Finally, a systematic approach to building and/or modifying the IFFS system in the future is included. The approach taken includes a "File Sorter" program which removes the need for much of the tedious work associated with building the system.
Implementation of the Graphics-oriented Interactive Finite Element Time-sharing System (GIFTS) on the PDP-11

by

John Trevor Sheldon
Lieutenant Commander, United States Navy
B. S., United States Naval Academy, 1967
M. S., Naval Postgraduate School, 1972

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1980

Author

Approved by: 

Thesis Advisor

Second Reader

Chairman, Department of Mechanical Engineering

Dean of Science and Engineering
ABSTRACT

The Graphics-oriented, Interactive, Finite Element Time-sharing System (GIFTS), written by Professor A. Kamel and Dr. Michael W. McCabe of the University of Arizona, has been implemented on the PDP-11 at the Naval Postgraduate School. This powerful system of programs was installed in a manner to facilitate its modification in the future. A very brief description of the GIFTS system, including the Unified Data Base, as well as the PDP-11 and RSX-11M operating system, are provided. Finally, a systematic approach to building and/or modifying the GIFTS system in the future is included. The approach taken includes a "File Sorter" program which removes the need for much of the tedious work associated with building the system.
TABLE OF CONTENTS

I. INTRODUCTION .............................................. 9
II. DESCRIPTION OF GIFTS ...................................... 11
    A. GIFTS DEVELOPERS ........................................... 12
    B. SYSTEM CAPABILITIES ........................................ 12
        1. Pre and Post Processing Capabilities ........ 14
            a. Pre Processing .................................. 14
            b. Post Processing ................................. 14
        2. Solving Capabilities .................................. 15
    C. THE COMPUTER PROGRAM .................................... 15
    D. THE UNIFIED DATA BASE .................................... 17
        1. Definition of Terms .................................. 17
        2. Naming Convention ................................... 19
    E. DOCUMENTATION OF GIFTS ................................... 20
        1. Reference Manuals .................................... 21
III. THE PDP-11 ................................................ 23
    A. ORGANIZATION ............................................. 23
    B. KH-11M OPERATING SYSTEM ................................ 23
        1. LOGON ................................................ 24
        2. User Identification Code ............................. 24
        3. Peripheral Interchange Program ...................... 24
        4. File Transfer Program ................................ 25
5. JOATAN Pour Plus .......................... 25
   a. /CO:20. .................................. 25
   b. /ER:HOME ................................. 25
6. Taskbuilder .................................. 25
7. Librarian Utility Program .................. 26
8. Text Editor .................................. 26
9. Macro Assembler ............................. 26
10. Execution Command .......................... 27
11. Use of Command Files ...................... 27

IV. THE BUILDING OF GITS ON THE POP-11 ...... 29
A. SOURC E TAP? ................................. 29
B. SORTING OF SOURC E LISTINGS ............... 31
   1. Description of TILSON ..................... 31
   2. TASKBUILDING GITS ....................... 34
      1. Non-Overlaid Modules ...................... 34
      2. Overlaid Modules .......................... 37
   D. BUILDING OF LIBRARIES .................... 38
   E. OVERLAP SCHEME USLDP .................... 39
   F. DELETION OF UNNecessary FILES ............ 40
   G. REBUILDING GITS .......................... 41

V. PROCEDURE FOR REVISION OF GITS ............. 42
A. MAKING MINOR CHANGES ....................... 42
   1. Changing the System Libraries .......... 42
   2. Changing a Module Library .............. 43
B. MAJOR CHANGES .............................. 44
C. UPDATING OF HELP FILE ..................... 45
VI. RECOMMENDATIONS ........................................... 46
APPENDIX A - DESCRIPTION OF THE GIFTS MODULES .......... 53
APPENDIX B - LIST OF GIFTS MANUALS ........................ 58
APPENDIX C - LISTING OF COMPUTER PROGRAM: FILSOR ...... 60
APPENDIX D - SAMPLE SESSION WITH FILSOR .................. 59
APPENDIX E - LISTING OF COMMAND FILE: BUILD.T.C.D ..... 73
APPENDIX F - LISTING OF GIFTS TAPES (NPS VERSIONS) ... 82
BIBLIOGRAPHY ..................................................... 83
INITIAL DISTRIBUTION LIST ..................................... 86
ACKNOWLEDGMENTS

I would like to take this opportunity to thank Professor Gilles Cantin for his patience and assistance, and Professor Paul Marto who, in Professor Cantin's absence, helped me through a major crisis. Also, there are not words to describe the patience shown by my family through my two tours at the Naval Postgraduate School. They deserve a lot more than merely mentioning their names, but notwithstanding this: Alice, Patricia, John and Peter.
I. INTRODUCTION

The purpose of this thesis is to describe the process whereby the Graphics-oriented Interactive Finite Element Time-sharing System (GIFTS) was implemented at the Naval Postgraduate School.

Anyone who has entered a problem with a large amount of numerical input into a computer knows the fear of making logical or typing errors which will go undetected. The GIFTS system goes a long way in reducing this problem by allowing a user to graphically reproduce the problem he/she has entered into the system. The solved problem can be displayed, as well, in a form which makes the effect of a given loading graphically reproducible at any time in the future.

The first step in building the system was obtaining the latest version (5.02) of the taped program from the Interactive Graphics Engineering Laboratory (IGEL) of the University of Arizona. After an initial attempt at "building" the system on the PDP-11 (using the methods described below) several minor errors were found. These errors were generally in the form of incomplete revisions and were easily correctable with telephone assistance from one of the developers, Mr. Michael W. McCabe, of the University of Arizona.
Since the average mechanical engineering student at the Naval Postgraduate School does not ordinarily spend time being taught on a computer system other than the school's mainframe (currently the IBM 360/67), a great deal of time was required in the preparation for this thesis simply learning the PDP-11/50 and the RSX-11M Operating System. Since it is expected that the GIFTS system will need to be revised in the future, it became obvious that an important objective of this thesis was to develop the means whereby changes to GIFTS could be made as conveniently and "painlessly" as possible without the need for a student or faculty member to become intimately familiar with the PDP-11. It is believed that this objective has been successfully met with the combination of a File Sorter program (FILSOR) and two "command" files. The net impact of these three programs is to allow a most powerful Finite Element Method (FEM) pre and post processor (plus solver) to be completely built on the PDP-11 with two tapes, two commands, and six hours of time.

It is believed that the GIFTS system will, in the future, be an invaluable teaching aid at the Naval Postgraduate School.
II. DESCRIPTION OF GIFTS

GIFTS is a system of programs used in solving Finite Element Problems. This statement does not really do justice to the system for the forte of the system is not in its ability to mathematically solve problems but rather in its ability to reliably and fairly completely define structural problems which are to be solved. It allows a user to:

1) Input problem parameters;
2) Observe the input both graphically and in tabulated form;
3) Update the model; and
4) Observe the output.

Many problems, due to their sizes, will be outside the range of the "solver" contained in the program. But, due to the highly structured nature of the Unified Data Base (UDB), other systems, more powerful in this area, can access the data, solve the problem, and return the solved problem to GIFTS for display.

The purpose of this section is to give enough of a description of GIFTS and the available documentation to assist a user interested in solving a FEM problem to find out how to get started at the Naval Postgraduate School.
A. GIFTS DEVELOPERS

GIFTS was written by Professor Hussein A. Kamel and Mr. Michael W. McCabe of the University of Arizona. The system is constantly being revised/updated as the need arises. The facility for expansion of the system is built in to it as not all element types have been implemented. As updates are received, they can be implemented by the procedures outlined below in section V.

B. SYSTEM CAPABILITIES

Much of the information included in this section is already included, in substance, in the "GIFTS Users' Manual." It is the purpose here to synthesize the information from this reference needed to have a general understanding of the system.

A list of the several program modules with descriptions can be found in Appendix A. Each has a purpose in formulating a FEM problem and more than one module is necessary to completely formulate a problem. However, not all program modules are necessary for every formulation.

The general breakdown of the module types/purposes is:
1) Model Generation and Editing;
2) Load and Boundary Conditions Generation, Display and Editing; and
3) General Purpose Computational and Result Display Modules.
In addition, there are modules available (but not yet implemented at the Naval Postgraduate School) to interface the GIFTS system with other Finite Element programs including:

1) ANSYS
2) SAP4
3) NASTRAN

The purpose of these interfaces is to act as "interpreters" of the GIFTS Unified Data Base in order that the generated model may be solved on one of these other systems. The interface program also takes the solutions generated by the other system and formats them back into the UDB for GIFTS in order that they can be displayed.

In the "GIFTS Users' Reference Manual," it is stated: "the generation and display capabilities of GIFTS go beyond its own analysis capabilities." It gives, by example, the fact that the GIFTS system can generate and display higher order elements while not (yet) being able to analyze the results. Though the author is not privy to a timetable, it is expected that the system capabilities will increase and can be added to existing capabilities currently at the Naval Postgraduate School. The methodology for making such revisions is covered below in chapter 7.
1. Pre and Post Processing Capabilities

a. Pre Processing

The GIFTS system is capable of being used as a pre-processor for other systems. It accepts commands which allow a user to establish the geometry of a model and to display it at a terminal for verification.

Figure 1 is an example of the program/user interaction which is required to establish the geometry of a plate with a hole in the center. Figure 2 is the resulting plot with elements and points labelled. Should an error be made during the session, a correction can be made before going on.

Also available to the user are a variety of tabulations of input and computed data. These also prove useful in the verification of a model.

b. Post Processing

Figure 3 depicts the results of the solved problem which was entered as in section 1.a. above. It depicts the stress contours as computed by (in this case) the GIFTS system for a given loading. If a different solver (e.g., SAP4) were to be used, the interface program would "translate" the output from the solver into the

---

1This problem is one that was included in the "GIFTS Primer" which was written at IGDL, University of Arizona. See Appendix 7.
GIPTS UDB format before using the GIPTS modules for displaying the results.

The system can also display deflection plots due to a given loading as well as computing and displaying time domain problems.

2. Solving Capabilities

The system, as presently configured, is capable of solving a wide class of structural, finite element method problems. However, there are some limitations. Page III-1 of the GIPTS User's Manual lists those elements which enjoy "Full Support" and, also, those within the categories: "Generation and Display Only" and "Storage Only." A user should be aware of these distinctions before deciding to solve a problem completely by GIPTS or by GIPTS in conjunction with another system.

3. THE COMPUTER PROGRAM

If loaded all at once, with no overlaying, the entire set of program/modules would take up to perhaps two mega bytes of memory. Since it is not desirable (or usually possible) to have this much space available to a user, the program has been divided into several, separately executable modules having as their common denominator the Unified Data Base.

Each program is called up (executed) by a "RUN" command. At the end of the session with an interactive module, a
"UNF" (or similar) command is given which causes the module to update the data base, close files and leave the module. To enter the next module another "UNF" command is given and so forth.

The interactive modules accept a large number of well-defined commands. Some of the modules have similar and even duplicative sub-objectives and therefore contain many of the same commands. Each program, however, has its own communications subroutine which will accept commands only valid in the particular module. Several different type prompt symbols are used (> , *, ?, blank) which make the nature of the input (i.e. alphanumeric, integer or floating point) less ambiguous.

As it was earlier stated, overlaying is required for most modules when installed on the PDP-11. This is due to a 64K byte limitation for any program segment. The overlaying schemes used for the several modules were included on the tapes received from the IGNE, University of Arizona, and are duplicated on the tapes discussed below in section IV.C.

One of the capabilities available to the user in many modules is that of plotting the model at a terminal. This feature obviously requires that a terminal with a graphics capability be used. The terminal for which the GIPTS System at the Naval Postgraduate School (NPS) is presently
geared is the Tektronix 4000 Series. To change to another type of terminal would require modifications to several of the GIPSS library subroutines.  

D. THE UNIFIED DATA BASE

During the course of model definition, the GIPSS system opens, performs input/output (I/O), and closes files on disc. At the end of a session one will find on his/her disc space several files having the jobname (specified by the user) as their filenames but each having a different "extension." These files represent the Unified Data Base (UDB).

The UDB files created by GIPSS are primarily random access, unformatted disc files. The fact they are unformatted makes storage of numerical data more efficient and the random access feature allows for easier identification of a particular record to be read or written.

1. Definition of Terms

The individual files are described in the "GIPSS Systems Manual" but a general description of the methodology of the programmers and the terminology used in the manual is warranted.

A "physical record," in context with the terminology used in the Systems Manual is the "collection of data" found

---

2 Specifically those in the file: MIT.5.PDP
in one record. When submitting a program written on punched cards, the input record length is limited to 30 - the number of characters allowed on the card. A disc, of course, does not have this limitation and the record size can be extremely large. (In the GIFTS system, the record size is automatically defined within the program and can be as large as 1634 bytes.) The program uses equivalent size buffers to accommodate the I/O record sizes and also allows for variable sizing of buffers/record size should the programs be run on a large machine. This fact is academic though since the current installation at the Naval Postgraduate School is on the PDP-11.

A "logical record" is a term used for the grouping together of data which are related insofar as the programmer says they are. Better put: "the smallest collection of data into which the data contained in a file may be divided." For example, a physical record of 200 bytes could be divided into ten logical records of 20 bytes each. In this case, the number "10" is the "blocking factor."

Data in GIFTS are generally buffered in named buffers. A buffer typically consists of a physical record plus some "bookkeeping" data. For example, a physical record in the "points" (PTS) file contains data on ten points. If the point being worked on by GIFTS is not in the record currently in the buffer, the current buffer is
stored in the PTS file and the correct record is read in. The "logical record" needed by GIFTS (i.e. the point being worked on) is now available.

2. Naming Convention

The UDB consists of, typically, ten or so files (the exact number depending on the problem being modeled). Certain administrative files are kept open throughout the life of a problem - that is, until deleted by the user. These files do not contain data which are used directly in solving or displaying a specific model. Other files are temporary or "scratch" files and are deleted prior to leaving the module which opened them. Then there are the files containing the data unique to a model which are "passed on" from module to module until the model/solution is completed. All of these files are the Unified Data Base - the focal point of the GIFTS system.

At the beginning of each module, the user is queried concerning the name of the model. The first time that this name is used (usually in the modules RUN or EDIT), the name becomes unique until the problem is deleted from the disc.\(^3\)

Figure 4 is a sample listing of the files built on GIFTS for the job "PLATE" which was shown in previous

\(^3\)This cannot be done by GIFTS but must be done with the file handler - see section III.1.3.
section. As the problem progresses, the number of files could increase and eventually take up a great deal of disc space (users should keep this in mind when creating a problem when disc space is at a premium).

Two other files exist which do not follow the naming convention which was outlined above. These are: \input{TS}.INF and \input{EST}.EST. The former is a sequential, formatted file listing all the "HELP" command answers that are available. It requires updating as changes are made to the system and is not tied to any particular problem. The latter file is used by OPTIM (i.e. optimization program) and is strictly for time estimates for the problem being completed. The user normally need not be concerned with this file, as it should already exist. If it doesn't, this will cause minor problems and could be easily rebuilt by running the module \input{ST.TSK} which is included on the magnetic tapes discussed below in section IV.3.

Each of the files is described in the GITS System Manual beginning on page S-11-2. For those interested in modifying the GITS system or writing an interface program, further explanation of the UDP is given below in section IV.

2. DOCUMENTATION OF GITS

The source listing as provided by the ISEI, University of Arizona, is liberally filled with comment statements. However, the interaction of the approximately 300 library
subroutines with the program and subroutines within the modules is so complex that trying to understand exactly what a program is doing at a particular time is virtually impossible without an excessive expenditure of time.

The user normally will not be interested in the source listing but rather in how to RUN the system. The remainder of this section is devoted to the documentation provided by the developers on how to use the system to solve a problem.

1. Reference Manuals

There are several manuals which are of interest to both the GIFTS user and the systems analyst responsible for building or maintaining GIFTS (see Appendix B). The manuals are provided with the GIFTS system by the University of Arizona and are kept at the Naval Postgraduate School in the Mechanical Engineering Department Computer Laboratory, Room 201D, Halligan Hall.

Two of these manuals have already been mentioned above: "The GIFTS Systems Manual"; and the "GIFTS User's Reference Manual." The former was used extensively in attempting to understand how the computer program worked and to understand the UDB. The latter was used in conjunction

---

4Not all the manuals have been provided by IGEL, University of Arizona, as they are yet to be written. For example, the "GIFTS System Installation Manual," which would have been useful here, has not yet been completed.
with the "GIFTS Primer" to obtain an understanding of how the system operated from the user's viewpoint.

The "Primer" serves as an excellent aid for the cautious, first-time user to get some hands-on experience with the system and to see what the system can actually do. It also explains, in detail, the purpose of several of the commands. The included examples, besides being educational for the first-time user, are very useful in checking the installation for accuracy.
III. THE PDP-11

The GIFTS system has been installed on the PDP-11/50, located in Room 500, Spanagel Hall, at the Naval Postgraduate School. The choice to install the system on this particular computer was based on its availability; the fact that GIFTS had already been brought up on the PDP-11 elsewhere; and, that the main computer system at NPS, the IBM 360, was being replaced in the near future.

There are actually two PDP-11s available in the Computer Lab: one with the UNIX operating system, and the other with RSX-11M. GIFTS was installed in the latter as it is limited to 32K work (64K byte) segments whereas UNIX allows only a 16K work (32K byte) segment size.

A. ORGANIZATION

The Computer Laboratory at the Naval Postgraduate School falls under the administrative control of the Director, Computer Laboratories. Under his/her control are several analysts/mathematicians familiar with the RSX-11M operating system.

B. RSX-11M OPERATING SYSTEM

The following are descriptions of utilities available under RSX-11M and which are used in the building and running
of GIFTS. The descriptions are provided here primarily as background information for section IV. The details of the following utilities may be found in the appropriate PDP-11 manuals.

1. **LOGON (HEL)**

"HEL" is the logon keyword. For the Mechanical Engineering Department, the logon is "HEL MEDEPT" whereupon the computer queries the user for an appropriate password. The logoff "keyword" is simply "BYE."

2. **User Identification Code (UIC)**

The UIC is assigned by the Director, Computer Laboratories, and serves two primary purposes in the RSX-11M operating system:

a. Identification of a particular user for security and accounting purposes; and

b. Identification of the user's directory on disc.

3. **Peripheral Interchange Program (PIP)**

PIP is the very versatile system of file handlers which is used to: move, delete, copy, rename, etc., files created on disc.

Some knowledge of PIP is essential to any prospective user of the GIFTS modules on the PDP-11. It allows for deleting and transferring files - which are useful "housekeeping" functions to know.
4. **File Transfer Program (FLX)**

FLX is the PDP-11 utility for handling files between disc and magnetic tapes.

5. **FORTRAN Four Plus (F4P)**

The FORTRAN compiler used to build the GIFTS system.

The syntax for this system allows for the use of many "switches." In the building of GIFTS on the PDP-11 it was only necessary to use two switches:

a. `/C0:20` - This switch was necessary on several of the subroutines to increase the number of allowed continuation cards from the default (i.e. 5).

b. `/TR:NONE` - This switch was used to build a separate system library which did not include the code necessary for tracing in the event of an object time error. This omission is necessary to allow the two largest modules to fit into the 32K word allowable segment on the PDP-11. (This will be further explained in section IV below.)

6. **Taskbuilder (TKB)**

TKB is the "linker" used under ASX-11. In conjunction with command files, it builds executable modules complete with overlays. A description of the command files used for building GIFTS is given in section IV. Further knowledge of the TKB utility would only be necessary if one were to rebuild or modify the GIFTS system without the help of the techniques which will be demonstrated in section V.
7. **Librarian Utility Program (LER)**

This utility is used to create and modify libraries of files. In the case of the building of the GIFTS system, it is used to create "system" and "module" libraries. In modifying the GIFTS system one would only need to become familiar with the syntax of two "switches": /IN = (that is, "insert"); and /DL: ("delete"). Examples are shown in section V.

3. **Text Editor (EDT)**

The RSX-11N system at the Naval Postgraduate School offers two text editors. "EDT" was selected because of its power. With a little imagination, a great deal of time can be saved in making major and/or repetitive changes to a file with EDT. To make future revisions to GIFTS, it is quite obvious that a knowledge of an editor would be necessary.

9. **Macro Assembler (.MAC)**

This is the keyword for assembling macro programs. For example, to assemble a program called TEST.MAC, one could enter:

```
MAC TEST = TEST
```

This would produce an object module called TEST. It is also possible to get a listing of the program with all external

---

5Note the syntax difference in the use of "=" for /IN and ":=" for /DL.
references, etc. For details concerning this and other features, a user should refer to the appropriate PDP-11 manual.

10. **Execution Order (RUN)**

RUN is the command under RSX-11M which causes an executable module to be loaded and executed. For example: RUN BULKM. For files that are overlaid, the executable module (with a default file extension of TSK) will need an additional file with the extension "STB". This is the "symbol table file" which is also built at the time of taskbuilding.

11. **Use of Command Files**

"Command" files are ASCII formatted files having an extension of "CMD." A Command file is executed by simply inserting the character "@" before the filename. For example, to run a command file called GIFT5.CMD, one would type in:

```
@GIFT5
```

The contents of the file would be executed line by line.

Another way in which command files can be used is in conjunction with a utility or the FORTRAN compiler, F4P. For example, if there are two separate FORTRAN programs to
be compiled, TEST1.MN and TEST2.MN, one could edit a command file called TEST1.CMD as follows:

```
>EDT TEST1.CMD
*I
TEST 1 = TEST 1
TEST 2 = TEST 2
{ctrl}Z
*exit
```

To execute this command file, one types:

```
F4P TEST1
```

This method can also be used for: PIP, TXE and LER.

\(^6\text{CTRL Z}\) is the combination of characters which allows the user to leave the "input mode" in EDT.
IV. THE BUILDING OF GIFTS ON THE PDP-11

The process of building GIFTS can be broken down into a few logical steps:

1) Sorting
2) Compiling
3) Library Building
4) System Building
5) Cleanup

To simplify some of these time consuming processes which must be completed, the author has written a "File Sorter" program (FILSOR) which effectively reduces the "slave labor" time and improves, it is believed, the accuracy of this process.

A. SOURCE TAPE

The PDP-11 version of the GIFTS system arrived on an unlabelled, ASCII-formatted, nine-track tape. Along with the tape was a listing of the names of the files and the sizes thereof. The files can be broken down by type as follows:

- Concatenated FORTRAN programs/subroutines: 29
- Overlay Description Files: 15
- Macro Programs: 2
- GIFTS Information File: 1
- Test Programs (FORTRAN): 3
The listings of FORTRAN programs/subroutines are unusable in the form they are received and must be separated, compiled, and the object code placed in libraries before the taskbuilding (linking) process can even begin. The steps that would be involved if this separation process were to be done manually with the Text Editor (EDT) are:

1. Find the first line of the program, subroutine or function; then
2. Find the last line of the program, subroutine or function (i.e. "END"); then
3. Write the inclusive lines between the first and last lines out to a new file; then
4. Go back to 1. until an IDF is reached.

The finding of the first and last lines using EDT is not difficult (except in each case, one must look for either a subroutine or a function since both occur). Writing to a new file is not particularly difficult but requires a rather lengthy line of commands. For example, to write lines 10130 through 13450, inclusive to a new file named FILEA/FEN, requires:

```
WR10130:13450/FILEA/FEN/UN
```

It can be imagined how long it would take to do this hundreds of times (about six hundred for GIFTS) without an error! For the reason that this task is so tedious and fraught with peril, the author wrote FILESOR.
B. SORTING OF SOURCE LISTING

1. Description of \textsc{filsor}

The basic \textsc{filsor} program accepts as input, the name of a source listing file\(^7\) containing at least two subroutines or one main program plus one or more subroutines (or functions). The following restrictions or guidelines concerning the use of \textsc{filsor} exist:

a. There are no "Block Data" subroutines within the source listing to be sorted;

b. If a listing contains a main program (vice a subroutine or a function), it must appear first in the file;

c. In all cases, the sorted program, subroutines and functions will have to be compiled;

d. In some cases, entire systems of sorted subroutines will have to be compiled with the "/TR:IOME" switch in use;

e. In all cases, the sorted and compiled subroutines will have to be stored in a library called, arbitrarily, \texttt{M1.OLE};

f. Comment cards preceding the first executable statement are discarded from the first output program;

---

\(^7\)The current version of the \textsc{gifts} source listings are on a magnetic tape in a format usable under the \texttt{FLX} utility of the \textsc{lsx-11} operating system (see above, section III.C.4). To obtain a listing of a particular magtape file, it would be necessary to load the file onto disc using \texttt{FLX} and then printing it using \texttt{PIP}.
The input listing is unaffected by FILSOR;

2. The "END" statement images must begin in column seven (otherwise the program will fail to recognize it as being the last statement in the program).

All the FORTRAN source listings included with GFTS conform to the above restrictions.

The main output from FILSOR is, of course, the separated FORTRAN files. The files are named according to the subroutine/function name or, in the case of a main program, the name of the input file. For example, assume a file named EXAMPLE.EXT contains: a main program and three subroutines (subroutines TEXT1, TEXT2 and Function TEXT3). The results of running EXAMPLE.EXT through FILSOR would be to create four new files called:

EXAMPLE.FTN
TEXT1.FTN
TEXT2.FTN
TEXT3.FTN

The original file EXAMPLE.EXT, containing the concatenated FORTRAN files, still exists and is unchanged by running FILSOR.

It should be emphasized that in the first program or subroutine in the file, blank or comment cards preceding the first executable statement will be "lost." Similarly, blank or comment cards between "END" and the next subroutine or function within the listing will be lost. Both these statements apply only to the sorted routines - not the original listing.
FILSCOR also builds two additional files while sorting the input file. Since it will eventually be necessary to compile all the subroutines, a file called LIB.CMD (a command file) is built which allows for the compilation of all program/subroutines sorted by FILSCOR.

As stated in section III.2.5, two possible combinations of "switches" occurred in the building of FILTS: one with the "/TR:NONE" switch, and one without. The "/NO:20" switch was used regardless. FILSCOR queries the user in order to determine whether he/she wants to include the trace capability or not.

The other file built by FILSCOR is called "STUFF.CMD." This file, in conjunction with the LBP utility, will store the object modules compiled with LIB.CMD into an object library called L1.0LB. After the "STUFF" process, the library L1.0LB can be renamed using PIP to avoid confusion with future FILSCOR operations.

Appendix C is a complete listing of FILSCOR. An example session is included in Appendix D.

A more complicated version of FILSCOR which allows a user to setup an input file with a list of several input files to be sorted is also included on magnetic tape. 9

9 This version is called FILS2 and requires an additional program, STUFF, to build the input file. Both of these modules are included on tape and are executed as a part of the automatic "BUILD" command file discussed below in section V and listed in Appendix E.
1. NON-OVERLAI D MODU LES

Most MMS modules must be overlaid on the FEP-11 in order that they can fit into memory. The syntax involved with the taskbuilder is quite extensively described in the FEP-11 manuals and will not be duplicated here. Several examples of the syntax will be shown and by these means the reader will be able to appreciate the methodology used on the FEP-11.

1. Non-Overlaid Modules

At present, there are seven modules which are not overlaid and, therefore, are the simplest to "build."

It is only necessary to compile these modules and taskbuild (building an executable module and "linking" with external references). To simplify the process even more, command files are normally used for the process and were used for building MMS.

For example, the module P.I.M. is built using the following command file:

```
P.I.M./ P/CP=P.I.M.,M.1:11/11

M1=16
M2=200
M10S=15
S=1:6
ADD=SY:7:1:3:0:11:12:13
ADD=SY:14:15
```

The switches used in the main line of the command file are necessary and more or less "boiler plate" switches.

34
are explained in detail in the PDP-11 manuals. The expression "MIFLIB/ ?" in the file indicates to the taskbuilder that besides the PDP-11 system library, (on the PDP-11, this is MSLIB.OLE and need not be referred to in the Command File as it is automatically called by TUP), a library called MIF LIB.OLE is called in order to resolve any external references. MIFLIB.OLE is actually made up of the compiled object modules from the following seven DASD tape files:

MIF1.D A
MIF2.D A
MIF3.D A
MIF4.D A
MIF5.D A
MIF6.D A
MIF7.D A

The other lines in the command file for MIF are worth mentioning for several of the MIF files is quite large (up to 1634 bytes) but since not all files are called by every module, the module will vary between modules. The size of MIFLIB for a particular module can be computed by referring to the table on page 1-7.2 of the TUP Systems manual.
"UNITs" defines the maximum number of the logical units whereas "ACTIVIL" assigns the number of active files which can be concurrently open. The latter is variable between modules and is quite important since the ACTIVIL parameter causes allocation of memory at the time of task-building. As many of the modules are quite tightly "packed" into the 327 word allowable memory segment, the extra bytes available by adjusting this parameter become very important.

"ASS" fulfills the taskbuilder requirement that each logical unit have assigned to it a physical unit. Thus, in the PRINT command file, logical unit six is assigned (ASS) to the terminal (TI:) and all "s" between seven and 15, inclusively, are assigned arbitrarily to the "system" disc (CY:).

Without the command file, each of the steps would have to be individually typed in. Since a command file was built in this case, it is merely necessary to type:

```
ok: print
```

It is worth noting here that if several modules are to be built, the command files may be interbed in another command file. For example, take the command file "V1 Programme" which is made up of:

```
the print
the savek
the residu
```

This file can be executed by typing: `V1`. 

2. Overlaid Modules

The majority of the GIMS modules are overlaid. Some of the overlay schemes are fairly complicated and are difficult, due to the taskbuilder syntax, to enter at a terminal. Therefore, as with the non-overlaid modules, command files are used. However, now "indirect" or "Overlay Description" files using "Overlay Description Language" (ODL) are also used. (These files are commonly referred to as "ODI files.")

The ODI file is built with the text editor for each module and describes the overlay scheme for the module. The file is then referred to by the UNIX command file. For example, the following is the command file used to build the module GMLIN in GIMS. Notice on the right hand side of the equal sign is the expression: FULK:/MP. The /MP switch indicates to UNIX that there exists a file on disk called "FULK.ODI" which describes the overlay scheme for the module (Figure 5). Note that no object modules are referred to directly in this command file:

```
FULK /MP/ODI, FULK=UNIX /.
/MP=13
/ODI=240
/MLC=15
/HL=11
/OH=7
/ID=10:11:12:13:14:15
//
```

The first line in the ODI file indicates the overlay scheme in symbolic terms (i.e., A, B, C, etc.). The other lines
indicate the choice of object modules for the "Root" and the various overlays. There are syntax and command rules which obviously must be followed in building an ODL file. Such information is found in: "RSX-11M Task Builder Reference Manual." It is not the purpose here to elaborate on this syntax.

The Command and ODL files for building GIFTS exist for all overlaid modules and are on magnetic tape for the eventuality that the system will need to be rebuilt. These files are the core of the work necessary for building GIFTS. Anyone interested in vastly revising GIFTS would need to know the existing structure of GIFTS and then attempt to reconstruct the effect of the revisions on the size of modules. As stated above, some of the modules are very tightly packed, some taking up to greater than 99 percent of the available 32K words.

1. BUILDING OF LIBRARIES

There are two basic types of libraries built from the GIFTS files. The first type includes the two separate system libraries. The reason for having a second "system" library is that two GIFTS modules, RULIB and RESULT, simply cannot fit into 32K words as normally built. Thus, a second nearly duplicate library is built using the "/BUILD" switch when compiling. The effect of this switch is to reduce the size of the object module by about ten percent.
The absence of the "trace" capability means that should an error occur during program run time, the system will not inform the user in which object module the error occurred. Again, this "problem" occurs only in TURBO and RESULT.

The other type of library is called a "module" library. That is, for every executable module where overlaying is being used, a library of the object modules derived from the individual program listing (vice the GIPS system library listings) is built. This approach allows the analyst to "keep track" of which object modules are needed for each overlaid module. Thus, this is a matter of convenience.

In Figure 5 are examples of how the two library types are used. Note that in every case where the switch "/LE" is seen, the preceding filename is the name of an object library. Where the switch "/LE" is used alone, as in:

IP31/LE, the meaning is that a check through the library GIPS/OLB will be made to resolve references. Where a colon is attached (i.e. "/LE:"), the NMR system will expect to find one or more specifically named object modules which are to be designated as being part of a particular segment.

E. OVERLAY SCHEMES USED

The magnetic tape received from NMS, University of Arizona, included the overlay schemes used at the Naval
Postgraduate School for the building of GITS. The schemes are actually in ODL file form. Changes to the overlay scheme(s) would be completed by making revisions to the respective ODL file and then rebuilding the respective module(s).

Installing the GITS system on another computer system could necessitate a revision to the schemes but the ODL files are a good point for departure.

F. DELETION OF UNNECESSARY FILES

Along the path of building GITS, one accumulates several files that are extraneous to the actual running of the GITS system. If file deletions are not completed, an accumulation of about 16,000 blocks of intelligence on disc (about twenty percent of the maximum capacity of the CDC 9762 disc drive) would be taken up by GITS. Since the executable module files accumulate to only about 4000 blocks, file deletions (using PIP) should be completed.

The method for doing this on the PDP-11 can be found in the appropriate PDP-11 Manual. Generally, it takes the form:

\[ \text{PIP Filename.Extension;Version/Deletion}\]

"Wild cards" are permitted for filenames, extensions and version names/numbers. The version number (or wildcard) must be included.
C. REBUILDING GFTS

... files necessary to rebuild GFTS exist on two magnetic tapes. A listing of the contents of the respective tapes are included as Appendix 7. To rebuild GFTS, it is merely necessary to load the tapes and type the following two commands:

```
PK /RS=.IT1: * BUILD2.CMD/DO
BUILD2
```

The resulting process takes approximately six hours to complete. A listing of BUILD2.CMD is included as Appendix 7.
V. PROCEDURE FOR REVISIGN GIFTS

A. MAKING MINOR CHANGES

It should be remembered that each module is listed separately. In addition, there are five files of subroutine listings plus two assembly language files which are included as part of the GIFTS system libraries (two). It should be quite obvious that if a revision to a single module listing is necessary then only that module will need to be rebuilt.

On the other hand, if one library subroutine is changed it would be wise to rebuild the entire system (unless the modules containing the revised subroutine can be isolated).

1. Changing the System Library

The following steps should be completed in revising the system libraries:

a. Edit (EDT) the listing (either LIB1.NEW, LIB2.NEW, LIB3.NEW, LIB4.NEW, or LIB5.PEP);

b. Extract the subroutine(s) which have been revised (in order that the entire library need not be rebuilt);

c. Compile the subroutine twice—once with the /CO:20 switch alone and again with the /TR:NONE switch;
d. Insert the object modules into the two libraries - GITLIB and GLIB2 by using the LER utility;

e. Rebuild the GIFTS system.

The last step is not quite as difficult as it seems since the command and ODL files are already built for this purpose. The entire rebuilding process can be done by a series of TKB "@" statements. Such a command file, called GIFTS5.CLD, is shown in Figure 6 and is included on the tapes mentioned in section IV.C. By merely typing @GIFTS5, the entire GIFTS system will be built in approximately one hour. The file depends, of course, on the existence of the command files, ODL files, GIFTS system libraries, and the respective module libraries to execute. A listing of the files needed to execute GIFTS5.CLD are shown in Figure 7.

2. Changing a Module Library

If only a single module listing is revised, then it should not be necessary to build the entire GIFTS system. In other words, the use of BUILD.CLD is unnecessary here. Instead, it would only be necessary to execute the steps which are demonstrated in Appendix D. The OPIM module is used by way of example in Appendix D, but any overlaid module would be "rebuilt" in the same manner.
For non-overlaid modules, it is necessary only to compile\textsuperscript{10} and taskbuild using the provided command files.

E. MAJOR CHANGES

If a substantial number of changes to the GPTS system were to be made, it may be necessary to rebuild the entire set of executable modules. Assuming that the command files are not to be revised,\textsuperscript{11} the following steps would be followed:

1. Revise the respective listing(s) using the Text Editor (EDT);
2. Revise the two tapes using FLX;
3. Execute @BUILDT.

It should be obvious that if the two existing tapes are to be modified that a new set of tapes will need to be built. The FLX utility is the handler for this process.

It should be noted that the present command file, BUILD.TXT, is based on the existence of two separate tapes with the contents being as listed in Appendix F. In this appendix,

\textsuperscript{10}It should be remembered that the default extension for a FORTRAN file on the PDP-11 is "FNT." The FORTRAN listings provided by IOLEL had the extension "NEW." Therefore, when compiling these programs using the P4P compiler, use syntax as follows (for the file called REDOS.NEW):

```
P4P REDOS=REDOS.NEW.
```

\textsuperscript{11}If new subroutine(s) were added to an individual module, then the respective "OIL File" would also need to be revised. It is also possible that changes to existing subroutines could make the individual module greater than 32K with existing overlay schemes. Then, a revised scheme may be necessary and the OIL file would have to be revised.
it should also be noted that the UIC for the tape file is: [20,1]. This UIC is presumed when RUIREDT is executed.

3. UPDATING OF HELP FILE

There exists a file called GIFTSS5.INT which contains the information or data used by the "HELP" command from the various GFTS modules. It will be necessary to use an editor to change this file. Revisions would be needed to this file only if updates were received from the University of Arizona.
VI. RECOMMENDATIONS

Several possibilities exist at the Naval Postgraduate School for the enhancement of the GIFTs system. A TEXTRONIX 4031 computer is already present within the Mechanical Engineering Department and could be used as an intelligent terminal. That is, it would be possible to operate with some of the GIFTs modules on a host computer such as the PDP-11 with the smaller modules being used independently on the 4031.

Of course, when the new mainframe replaces the currently used IBM 360/67 in FY 1981, a worthwhile project would be to install GIFTs on it.

In addition, it is recommended that the interface program for the SAP4 system, which is currently available at NPS, be obtained from IGML, University of Arizona, in order that the SAP4 and GIFTs can be "tied together."
BULKM VER. 5.02

TYPE JOB NAME
PLATE JOB PLATE BEING CREATED
* MSTEEL
   > 1
   >
* ETH,1
   > 1
? 0.1
>
* KPOINT
   > 1
? 2,,
> 2
? 2,,
> 3
? .8,
> 4
? 6,6,
> 5
? 6,,
> 6
? 1.414,1.414,
>
* SLINE
   > L23
? 2,3,5
> L15
? 1,5,5
> L34
? 3,4,4
> L45
? 5,4,4
>
* CARC
> C12
? 1,6,2,7
>
* COMPLINE
> L35
? L34,L45
>
* GETY
> QM1
? 1,1
>
* GRID4
> QUARTER
? C12,L23,L35,L15
>
* KN
* LNAM
* GNAM
* PLOT

Figure 1 - Program Interaction for PLATE
Figure 2 - GIFT5 Presentation of PLATE Input
>PIP PLATE */L1

DIRECTORY DP3 [20.1]
23-AUG-80 13:17

PLATE:FIL:1 1 23-AUG-80 12:07
PLATE:MAT:1 1 23-AUG-80 12:07
PLATE:THS:1 1 23-AUG-80 12:07
PLATE:PTS:1 9 23-AUG-80 12:19
PLATE:ELT:1 18 23-AUG-80 12:07
PLATE:LIN:1 10 23-AUG-80 12:07
PLATE:GRD:1 7 23-AUG-80 12:07
PLATE:PAR:1 1 23-AUG-80 12:15
PLATE:LOS:1 7 23-AUG-80 12:20
PLATE:ELD:1 14 23-AUG-80 12:20
PLATE:SDY:1 1 23-AUG-80 12:27
PLATE:STF:1 23 23-AUG-80 12:28
PLATE:LDI:1 6 23-AUG-80 12:30
PLATE:CN1:1 6 23-AUG-80 12:30
PLATE:DNS:1 5 23-AUG-80 12:30
PLATE:STR:1 7 23-AUG-80 12:31

TOTAL OF 117 /160 BLOCKS IN 16 FILES

Figure 4 - Listing of Files Created by GIFTS JOB PLATE
BRO TI: BUILDING GIFTS

TKB @STRESS
TIME
TKB @EDITLB1
TIME
TKB @SAVER
TIME
TKB @RESIDU
TIME
TKB @BULKLB
TIME
TKB @STIFF
TIME
TKB @BULKM
TIME
TKB @DEFL
TIME
TKB @RESULT
TIME
TKB @BULKF
TIME
TKB @DECOM
TIME
TKB @EDITM
TIME
TKB @EDIT
TIME
TKB @AUTOL
TIME
TKB @DEFCS
TIME
TKB @LOCAL
TIME
TKB @SUBS
TIME
TKB @TRAN1
TIME
TKB @TRAN
TIME
TKB @OPTIM
TIME
TKB @PRINT
TIME
TKB @TRAN2
TIME
TKB @BRECS

Figure 6  Listing of GIFTS.S.CMD

Figure 7  Listing of Files Needed to Execute GIFTS.S.CMD
APPENDIX A

DESCRIPTION OF GIFTS MODULES

MODEL GENERATION AND EDITING

BULKX

BULKX is an automated three dimensional plate and shell model generator. It is suitable for large continuous structure that can be easily modeled by repetitious generation of points and elements.

EDITX

EDITX is designed to update and correct BULKX models, although it can be used to generate simple models and ones too complex for BULKX.

EDIT3

EDIT3 accepts information regarding external and dependent boundary nodes in a constrained substructure.

BULKS

BULKS is a three dimensional solid model generator. One may ask for the display of the edges, and may add and display selected point and element slices.

---

12 The descriptions here are taken from the "GIFTS User's Manual."

13 BULKS, as of the date of this writing, is not yet implemented on the PDP-11. This is primarily due to its size.
LOAD AND BOUNDARY CONDITION
GENERATION, DISPLAY AND EDITING

BULKF

BULKF is intended to allow only those freedoms which a model can support, thereby relieving the user of the necessity of suppressing all superfluous freedoms by hand.

BULKLD

BULKLD is a bulk load and boundary condition generator designed to apply load to models generated with BULKF. It may be used to apply distributed line and surface loads and masses, prescribed displacements along lines and surfaces and inertial loads. Temperatures may also be applied to lines and surfaces.

EDITL

EDITL is a display and edit routine intended to provide local modification capability to loads and boundary conditions applied by BULKLD. It may also be used to generate simple loading on models, or loading on models not generated with BULKF. Temperatures may also be applied to elements. After BULKLD has been run, the thermal and combined loads may be examined.

LOADS

LOADS is a load and boundary condition generator for solid models. Loads may be distributed on lines or

\footnote{LOADS, as of the date of this writing, is not yet implemented on the PDP-11. This is primarily due to size.}
surfaces. Loads and boundary conditions may be displayed on point slices.

GENERAL PURPOSE COMPUTATIONAL
AND RESULT DISPLAY MODULES

OPTIN

OPTIN is a bandwidth optimization program. Although GIFTS is designed to handle problems without size or bandwidth restrictions, it is very important that the problem be optimized before the solution proceeds. Experience has shown that run times can be reduced by a factor of two to ten if the procedure is used. OPTIN may be called several times in a row, until the best node numbering scheme has been achieved.

STIFF

STIFF performs computation of the stiffness matrices and assembles them into the master stiffness matrix.

BECO:

BECO introduces kinematic boundary conditions, and decomposes that stiffness matrix by the Cholesky method.

DIC3:

DIC3 computes the deflections from the current loading conditions and the decomposed stiffness matrix. If temperatures are present, thermal forces will be calculated and added to the current applied loads before solution.
STRESS

STRESS computes the element stresses based on the current deflections.

RESULT

RESULT displays deflections and stresses. It has many options that may be used, at the discretion of the user, to transform the results for optimum comprehension.

THE SIES NATURAL VIBRATION PACKAGE

AUTOL

AUTOL is ordinarily used to generate starting loads for the subspace iteration to compute natural modes of vibration.

SUBS

SUBS performs a single subspace iteration to determine the model's natural modes. It must be repeated as many times as necessary to obtain convergence to the desired extent.

THE SIES TRANSIENT RESPONSE PACKAGE

(DIRECT INTEGRATION)

TRAN1

TRAN1 is to be run on a transient response model immediately after stiffness assembly. It is used to specify the time step to be used in the integration process.
TRAN2

TRAN2 is run after TRAN1 and DECOM. It computes the displacement matrix for time T.

TRANS

TRANS maintains and plots histograms of the displacements of up to four different freedoms.

GIFTS CONSTRAINED SUBSTRUCTURING PACKAGE

REDCS

Before a COSUB module may be used in a master analysis run, it must be preceded by program REDCS to form a reduced stiffness matrix and a reduced load matrix (if there are any loads associated with the COSUB).
APPENDIX E

LIST OF GIFTS MANUALS

GIFTS USER'S REFERENCE MANUAL

"Contains complete and detailed description of all GIFTS commands and computational procedures. It is meant as a source of information for the experienced user."

GIFTS SYSTEMS MANUAL

"Contains detailed information on the code, data base and program structure. Useful for those undertaking program conversion or enhancement."

Though there is a great deal of detail concerning the UDD and program structure (for the ECLIPSE Computer), there is really insufficient information to get started in a "program conversion or enhancement." There are several terms and acronyms which are undefined in the description where knowledge of the other manuals are essential for understanding.

GIFTS PRIMER

"... useful to new users, and to exercise the system on a new installation, or check out a new version of the program on an existing installation ... Tutorial ... Solved Examples."

This manual is excellent for the intended purposes. Anyone seriously intending to use the system should spend several hours with this manual and the computer.

Remarks in quotation marks are taken directly from pages 337-338 and 4 from the GIFTS Primer.
GIFFTS INSTALLATION MANUAL

"Designed to help those attempting to install the program on their own system. Describes implementation and test procedures."

This manual, as of this writing, is not implemented. It is hoped that with respect to the PDP-11, this thesis provides some of the information needed.

GIFFTS THEORETICAL MANUAL

"Contains mathematical fundamentals and algorithms underlying mesh generation, element characteristics and solution procedures. Of use to those wishing to assess the properties of the mathematical model used, or modify the program."

GIFFTS MODELLING GUIDE

"Aimed at the program user. Discusses practical aspects of finite element modelling in general and pays particular attention to elements and procedures implemented in the GIFFTS system."

Not implemented as of this writing.

GIFFTS POCKET MANUAL

"A handy pocket-size reference manual containing complete, but terse, summary of information in the GIFFTS Users Reference Manual. Used mainly as a quick reference manual to be used while working on a terminal by the experienced user."

Emphasize experienced!
APPENDIX C

LISTING OF PROGRAM: FILSOR

C**************************FILE SORTER**************************
C**********WRITTEN BY JOHN T. SHELDON**********
C*****************************************************************
*** LAST UPDATED 8/25/80 BY JTS

THIS PROGRAM IS A FILE SORTER.  IT WAS SPECIFICALLY
WRITTEN WITH THE "GIFS" SYSTEM IN MIND.  THE GIFS
SOURCE LISTINGS HAVE THE FOLLOWING CHARACTERISTICS:

1) THERE ARE NO BLOCK DATA SUBROUTINES;

2) SOME OF THE CONCATENATED LISTINGS START
   WITH A MAIN PROGRAM;

3) FIVE OF THE FORTRAN LISTINGS INCLUDE ONLY
   SUBROUTINES OR FUNCTIONS (IE NO MAIN
   PROGRAMS);

4) IN ALL CASES, THE PROGRAM, SUBROUTINES AND
   SUBROUTINES(PSF) WILL HAVE TO BE COMPILED;

5) IN SOME CASES, ENTIRE SYSTEMS OF SORTED
   SUBROUTINES WILL HAVE TO BE COMPILED
   WITH THE "/TR:NONE" SWITCH IN USE;

6) IN ALL CASES, THE SORTED PSF'S WILL HAVE
   TO BE PUT IN A LIBRARY.

7) THE COMMENT STATEMENTS PRECEDING THE FIRST
   EXECUTABLE STATEMENT ARE DISCARDED;

8) THE INPUT FILE IS UNAFFECTED BY THIS PROGRAM;

9) THE "END" STATEMENT BEGINS IN COLUMN 7

BYTE YES,NO,ANS,ANANS(40),LINE(72),OBLANK
BYTE F,E,NN,C,S,DD,SFLAG

00
000030
000040
000050
000060
000070
000080
000090
001000
001100
001200
001300
001400
001500
001600
001700
001800
001900
002000
002100
002200
002300
002400
002500
002600
002700
002800
002900
003000
003100
003200
003300
003400
003500
003600
003700
003800
003900
004000
004100
004200
004300
LOGICAL OK, ENDIT
COMMON SFLAG
DATA F,N,DD /1HF,1HE,1HN,1HD/
1 IEND;/FLAG,SFLAG,OBLANK/0,0,0,1H/
2 YES,N,F,C,S/IHY,1HN,1HC,1HS/
IFTRACE=0
C
WRITE(6,100)
WRITE(6,106)
C
IF ((I-END) EQ.0) THEN STOP
ENDIF
C
WRITE(6,104)
READ(5,101) ANANS(I)
IF (ANANS(I).EQ.Y) ITTRACE=1
IF (ITRACE.EQ.1) SFLAG=1
C
GET NAME OF FILE TO BE SORTED
WRITE(6,104)
READ(5,101) ANANS
C
OPEN STATEMENTS REQUIRES THAT LAST BYTE IN NAME BE 0
ANANS(40)=0
C
OPEN INPUT FILE
OPEN(UNIT=1,NAME=ANANS,TYPE='OLD',ACCESS='SEQUENTIAL',
1 DISP='SAVE')
C
LOOKING FOR FIRST EXECUTABLE STMT IN PROGRAM/SUBROUTINE OR
FUNCTION(PSF)
C
15 CALL RDLINE(LINE)
C
"N" EQUALS THE NUMBER OF LAST NON-BLANK CHARACTER
C
IN THE LINE
N=0
DO 17 I=1,72
17 IF (LINE(I).NE.,BLANK) N=1
C
IF LINE IS A COMMENT CARD, IGNORE IT
IF (LINE(I).EQ.,GO TO 15
C
BLANK LINE?
IF (N.EQ.0) GO TO 15
C
BEGINNING OF PSF(HOPEFULLY)
C
GET NAME OF FILE
CALL NAMEFL(LINE,ANANS)
C
PREPARE AND STORE COMMAND FILE INPUTS
CALL CMDFIL(ANANS)
ANANS(40)=0
C
OPEN OUTFILE
OPEN(UNIT=2,NAME=ANANS,TYPE='NEW',ACCESS='SEQUENTIAL',
1 DISP='SAVE')
C
START STORING AND READING UNTIL END STATEMENT ENCOUNTERED
ENDIT=.FALSE.
20 WRITE(2,101)(LINE(I),I=1,N)
IF(EDIT.EQ.,TRUE.)GO TO 25
CALL RDLINE(LINE)
C LOOK FOR "E" IN COLUMN 7 AND "N" IN COLUMN 8
ENDIT=(LINE(7).EQ.4) .AND. (LINE(8).EQ.N)
C MAKE OTHER CHECKS TO ENSURE THIS IS AN END STATEMENT
IF(.NOT.(ENDIT .AND. (LINE(9).EQ.0D .AND. LINE(10).EQ.32)))
ENDIT=.FALSE.
IF(LINE(1).EQ.C)ENDIT=.FALSE.
N=0
DO 21 I=1,72
21 IF(LINE(I).NE.OBLANK)N=I
GO TO 20
C
DONE WITH THIS OUTFILE
25 CLOSE(UNIT=2,DISP='SAVE')
GO TO 15
100 FORMAT (20X,'***FILE SORTER***',/
101 FORMAT(80A1)
104 FORMAT('TYPE IN NAME OF FILE INCLUDING EXTENSION:')
105 FORMAT(1X,80A1)
106 FORMAT('"TR:_NONE" SWITCH DESIRED FOR COMPIL? (Y OR N):')
END
SUBROUTINE CMD(FILNAM)
******************************************************************************
******************************************************************************
*********WRITTEN BY JOHN T. SHELDON*************
******************************************************************************
*** LAST UPDATED ON 8/25/80 BY JTS
******************************************************************************
********* THIS SUBROUTINE IS PART OF THE FILSOR PACKAGE
******************************************************************************
LIB.CMD AND STUFF.CMD INPUTS.
******************************************************************************
LIB.CMD IS A FILE OF STATEMENTS WHICH ARE
USED ALONG WITH THE F4P COMPILER ON THE
POP11. THE OUTPUT LINES ARE OF THE FORM:
/TR:NONE
THE /CO:20 SWITCH HAS BEEN INSERTED AS
A FEW OF THE SUBROUTINES WILL HAVE MORE
THAN THE DEFAULT CONTINUATION LINES.
THE /TR:NONE SWITCH IS AN OPTION WHICH
IS USED IF SPACE IS AT A PREMIUM (AS IN
THE RESULT AND BULKLB MODULES OF GIFTS.
******************************************************************************
STUFF.CMD IS A FILE OF INDIVIDUAL "LBR" COMMANDS WHICH "STUFF" THE COMPILED OBJECT MODULES IN A LIBRARY CALLED (ARBITRARILY): L1.LDLB

THE ABOVE TWO FILES ARE EXECUTED IN THE FOLLOWING ORDER AND WITH THE SAME SYNTAX:

@F4PLIB
@STUFF

ERRORS DURING Compile WOULD BE PRINTED, OTHERWISE NO OUTPUT SHOULD BE EXPECTED FROM THE FIRST LINE.
STUFF, ON THE OTHERHAND, WILL PRINT EACH COMMAND LINE.

BYTE ISLASH,IEQ,SFLAG
BYTE FILNAM(40),OBLANK,LIBNAM(2),IDOT,LOUTPT(40),MOUTPT(40)
DIMENSION LOUTP(20),MDUP(5),LLOUP(7)
COMMON SFLAG
EQUVALENCE (LOUTPT(1),LLOUP(1)),(MOUTPT(1),MDUP(1))
DATA IEQ/IH=/
DATA LLOUP(1),MDUP(2),MDUP(1),MDUP(2)/2H,2H,2HLB,2HR /
1 IFLAG,LIBNAM(1),LIBNAM(2),OBLANK/O,1HL,1H1,1H /
2 IFLAG,MDUP(3),MDUP(4),MDUP(5) /1H.,2HL1,2H/1,2HN /
3 LLOUP(1),LLOUP(2),LLOUP(3)/2H/C,2HO,2H20 /
4 LLOUP(4),LLOUP(5),LLOUP(6)/2H/T,2HR,2HNO /
5 LLOUP(7)/2HNE/
SFLAG = 1 INDICATES /TR:NONE Switch IS TO BE LEFT IN
SFLAG = 0 WILL CAUSE THE ARRAY HOLDING THIS EXPRESSION
TO BE ZERSED
IF(SFLAG.EQ.1)GO TO 5
DO 3 I=4,7
3 LLOUP(1)=0
SFLAG=0

CHECK TO SEE IF FIRST PASS THROUGH SUBROJTIME
IF IT IS, MUST OPEN FILES
5 IF(I1FLAG.EQ.1)GO TO 10

OPEN COMMAND FILES
LU=3 "STUFF.CMD" FILE
OPEN(UNIT=3,NAME='STUFF.CMD',TYPE='UNKNOWN',
1 ACCESS='APPEND',DISP='SAVE')
C  LU=4  "LIB.COM"  FILE
10  OPEN(UNIT=4,NAME="LIB.COM",TYPE='UNKNOWN',
     ACCESS='APPEND',DISP='SAVE')
10  IFLAG=1
10  DO 15  I=1,6
      IF(FILNAM(I).EQ.OBLANK.OR.FILNAM(I).EQ.IDOT)GO TO 16
10   N IS THE NUMBER OF CHARACTERS IN THE NAME OF THE PSF
10   N=1
15  CONTINUE
 C  M,N---
 C  OF CHARACTERS IN "FILNAM"
16  M=N
 C  BUILD LIB.COM FILE INPUT
   N=N+4
20  LOUTPT(I)=FILNAM(I-4)
   N=N+1
   LOUTPT(N)=IEQ
   DO 30  I=N+1,N+4
30  LOUTPT(I)=FILNAM(I-M-5)
   DO 35  I=N+M+1,26
35  LOUTPT(I)=OBLANK
   DO 36  I=14,20
36  LOUTPT(I)=LOUP(I-13)
   WRITE(4,100)LOUTPT
 C  BUILD STJFF.COM FILE INPUT
   DO 40  I=1,N+10
40  MOUTPT(I)=FILNAM(I-10)
   DO 45  I=M+11,43
45  MOUTPT(I)=OBLANK
   WRITE(3,100)MOUTPT
   RETURN
100  FORMAT(80A1)
END
 C***************SUBROUTINE NAMEFL(LINE,ANAVS)
 C***************WRITTEN BY JOHN T. SHELDON***************
 C***************LAST UPDATED ON 8/25/80 BY JTS
 C***************THIS SUBROUTINE IS A PART OF THE FILSOR PACKAGE.
 C ITS PURPOSE IS TO OBTAIN THE NAME OF THE "OUTFILE"
BE IT A MAIN PROGRAM, SUBROUTINE OR FUNCTION.

A LINE OF DATA IS ENTERED(LINE) AND THE PSF IS
RETURNED(ANANS).
BYTE LINE(72), ANANS(40), S(12), F(8), OBLANK, PAREN
BYTE IDOT, FF, TT, NN, SFLAG
COMMON SFLAG
DATA S(1), S(2), S(3), S(4), S(5) /HS, IU, IH, IH, IH, IH/
1 S(6), S(7), S(8), S(9), S(10), IU, IH, IH, IH, IH/
2 F(1), F(2), F(3), F(4), F(5) /IH, IH, IH, IH, IH/
3 F(6), F(7), F(8), OBLANK, PAREN /IH, IH, IH, IH, IH/
4 FF, TT, NN, IDOT, OFLAG /IH, IH, IH, IH, IH/.

ZERO ANANS
IF(0, OFLAG, EQ, 0) GO TO 11
DO 10 I=1, 40
10 ANANS(I) = OBLANK
CONTINUE

IFLAG=1

LOOK FOR "SUBROUTINE" OR "FUNCTION"
DO 15 I=7, 50
N=I

LOOKING FOR LETTER "S"
IF(Line(I). EQ. 'S') GO TO 20
DIDN'T FIND S, HOW ABOUT AN "F"
15 IF(Line(I). EQ. 'F') GO TO 30

THIS MUST BE A MAIN PROGRAM SINCE THERE IS NO LETTER
"S" OR "F" IN THE FIRST EXECUTABLE LINE.
CONTINUE

THIS MEANS THAT "ANSWER"(ANANS) CONTAINS
RIGHT FILE NAME BUT WRONG EXTENSION
(THE NAME OF THE MAIN PROGRAM IN THE GITS
LISTINGS IS ALWAYS THE FIRST IN THE PACKAGE.
THEREFORE, TO HAVE GOTTEN TO THIS POINT, WE'RE
LOOKING FOR A MAIN PROGRAM. ANANS HASN'T BEEN
CHANGED SINCE IT WAS USED TO GET THE NAME OF
THE INPUT FILE.
DO 17 I=1, 6
17 CONTINUE

IF(ANANS(I). EQ. IDOT) GO TO 18
II=I
IF(ANANS(I). EQ. IDOT) GO TO 18
18 CONTINUE
II=II+1
19 ANANS(II+1) = FF
ANANS(II+2) = TT
ANANS(II+3) = NN
ANANS(40) = 0
RETURN

CHECK IF A "SUBROUTINE". IF NOT THEN MUST
C     BE THE BEGINNING OF THE MAIN PROGRAM
20  IDIF=N-1
   DO 21 I=N,N+9
21  IF(LINE(I).NE.S(I-IDIF))GO TO 16
   C     MUST BE A SUBROUTINE
   C     FIRST ZERO ANANS
   DO 22 I=1,40
22  ANANS(I)=0
   N=N+10
   DO 23 I=N,N+5
23  IF(LINE(I).NE.CBLANK)GO TO 24
24  N=N+1
   IDIF=N-1
   DO 25 I=N,N+6
25  IF(LINE(I).EQ.CBLANK.OR.LINE(I).EQ.PAREN)GO TO 26
26  ANANS(I-IDIF)=LINE(I)
   ANANS(I-IDIF+1)=FF
   ANANS(I-IDIF+2)=TT
   ANANS(I-IDIF+3)=NN
   ANANS(40)=0
   RETURN
   C     CHECK IF A "FUNCTION". IF NOT, MUST THE BEGINNING OF A PROGRAM
   C     IDIF=N-1
   DO 31 I=N,N+7
31  IF(LINE(I).NE.F(I-IDIF))GO TO 16
   C     MUST BE A FUNCTION
   C     FIRST ZERO ANANS
   DO 32 I=1,40
32  ANANS(I)=0
   N=N+8
   DO 33 I=N,N+5
33  IF(LINE(I).NE.CBLANK)GO TO 34
34  N=N+1
   IDIF=N-1
   DO 35 I=N,N+6
35  IF(LINE(I).EQ.CBLANK.OR.LINE(I).EQ.PAREN)GO TO 26
   ANANS(I-IDIF)=LINE(I)
   GO TO 26
102  FORMAT(80A1)
   END
   SUBROUTINE RLINE(LINE)
C*****************************************************************************
C*****************************************************************************
C     WRITTEN BY JOHN T. SHELDON***********
C*****************************************************************************
C     LAST UPDATED ON 8/25/80 BY JTS
***********************************************************************
THIS SUBROUTINE IS PART OF THE FILSOR PACKAGE

THE PURPOSE OF THIS SUBROUTINE IS TO READ A LINE FROM THE INPUT FILE. IF AN "EOF" IS
REACHED, ALL OPEN FILES ARE CLOSED AND THE FILSOR PROGRAM IS STOPPED HERE.

BYTE LINE(72)
READ(I,100,END=10)LINE
RETURN
10 DO 20 I=1,4
20 CLOSE(UNIT=1)
STOP
100 FORMAT(80A1)
END
*********************************************************************
APPENDIX D

SAMPLE SESSION WITH HILSIR

The following is a listing from an actual run with HILSIR. It has been annotated to indicate what actually is going on and the reasons for the various steps.

The HILSIR3.01 file executes this entire process "automatically" with the exception that the HILSIR2 version of HILSIR is needed as well as the STUDENT file (which generates the answers to the HILSIR questions asked below). HILSIR, of course, also reads the necessary files from magnetic tape.
>PIP/LI

DIRECTORY DP3 [160,53] (1)
30-AUG-80 15:46

OPTIM.ODL 147 1 30-AUG-80 15:33
FILSOR.TSK 12 51 15:33
OPTIM.NEW 15:33
OPTIM.CMD 10 1 15:35
CIFTLIB.OLB 1 592 15:45

TOTAL OF 698/698 BLOCKS IN 5 FILES

>RUN FILSOR (2)

***FILE SORTER***

"/TR: NONE" SWITCH DESIRED FOR COMPIL? (Y OR N) N
TYPE IN NAME OF FILE INCLUDING EXTENSION OPTIM NEW

>FXP.OLIB (3)

>PIP *.OBJ/LI (4)

DIRECTORY DP3 [160,53] (4)
30-AUG-80 15:48

OPTIM.OBJ 1 30-AUG-80 15:46
BAND.OBJ 6 30-AUG-80 15:47
INOPT.OBJ 1 30-AUG-80 15:47
OPT.OBJ 1 30-AUG-80 15:47
SWAP.OBJ 3 30-AUG-80 15:47
TEROPT.OBJ 1 30-AUG-80 15:47

TOTAL OF 39/50 BLOCKS IN 6 FILES (5)

>LBR L1/CR 39 6.6.OBJ (6)
>
>
>
>
>

>STUFF (7)

>LBR L1/IN=OPTIM
>LBR L1/IN=BAND
>LBR L1/IN=INOPT
>LBR L1/IN=OPT
>LBR L1/IN=SWAP
>LBR L1/IN=TEROPT
>
>
>
>
>

>EOF

>PIP OPTIM.OLB-L1.OLB/RE (8)

>TKB OPTIM (9)
<table>
<thead>
<tr>
<th>File</th>
<th>Date</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUFF.CMD,1</td>
<td>30-AUG-80</td>
<td>1</td>
</tr>
<tr>
<td>OPTIM.OOL,1</td>
<td>30-AUG-80</td>
<td>1</td>
</tr>
<tr>
<td>LIB.CMD,1</td>
<td>30-AUG-80</td>
<td>1</td>
</tr>
<tr>
<td>FLSOR.TSK,2</td>
<td>30-AUG-80</td>
<td>53</td>
</tr>
<tr>
<td>OPTIM.NEW,1</td>
<td>30-AUG-80</td>
<td>2</td>
</tr>
<tr>
<td>OPTIM.FTN,1</td>
<td>30-AUG-80</td>
<td>10</td>
</tr>
<tr>
<td>OPTIM.CMD,10</td>
<td>30-AUG-80</td>
<td>7</td>
</tr>
<tr>
<td>BAND.FTN,1</td>
<td>30-AUG-80</td>
<td>7</td>
</tr>
<tr>
<td>INOPT.FTN,1</td>
<td>30-AUG-80</td>
<td>18</td>
</tr>
<tr>
<td>OPT.FTN,1</td>
<td>30-AUG-80</td>
<td>4</td>
</tr>
<tr>
<td>SWAP.FTN,1</td>
<td>30-AUG-80</td>
<td>14</td>
</tr>
<tr>
<td>TEROPT.FTN,1</td>
<td>30-AUG-80</td>
<td>1</td>
</tr>
<tr>
<td>OPTIM.OBJ,1</td>
<td>30-AUG-80</td>
<td>6</td>
</tr>
<tr>
<td>INOPT.OBJ,1</td>
<td>30-AUG-80</td>
<td>5</td>
</tr>
<tr>
<td>OPT.OBJ,1</td>
<td>30-AUG-80</td>
<td>10</td>
</tr>
<tr>
<td>SWAP.OBJ,1</td>
<td>30-AUG-80</td>
<td>14</td>
</tr>
<tr>
<td>TEROPT.OBJ,1</td>
<td>30-AUG-80</td>
<td>3</td>
</tr>
<tr>
<td>GIFTLIB.OLB,1</td>
<td>30-AUG-80</td>
<td>592</td>
</tr>
<tr>
<td>OPTIM.OLB,1</td>
<td>30-AUG-80</td>
<td>10</td>
</tr>
<tr>
<td>OPTIM.TSK,1</td>
<td>30-AUG-80</td>
<td>209</td>
</tr>
<tr>
<td>OPTIM.STB,1</td>
<td>30-AUG-80</td>
<td>6</td>
</tr>
</tbody>
</table>

TOTAL OF 1846 /1086 BLOCKS IN 22 FILES

> PIP * FTN,*/DE
> PIP * OBJ,*/DE
> PIP * CMD,*/DE
> PIP * OOL,*/DE
> PIP * OLB,*/DE
> PIP FLSOR.TSK,*/DE
> PIP OPTIM.NEW,*/DE
> PIP/LI

<table>
<thead>
<tr>
<th>File</th>
<th>Date</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIM.TSK,1</td>
<td>30-AUG-80</td>
<td>209</td>
</tr>
<tr>
<td>OPTIM.STB,1</td>
<td>30-AUG-80</td>
<td>6</td>
</tr>
</tbody>
</table>

TOTAL OF 215 /219 BLOCKS IN 2 FILES
(1) The first thing done is a "PII/UI" which lists all files in the directory. In this case, the response indicates that we're in directory 150,53. The files listed are all the files needed to build OPIIUI. If RESULT or RESULT were being built, then GLIB2 would replace RPIILIB as the LILIB "System" library. Also, as a means of differentiating the LUMIB and RESULT module libraries (see section L7.C) these libraries are arbitrarily referred to, in the command files, as LUMIL and RESULT, respectively.

(2) TILSOS is executed. It responds by asking two questions before proceeding.

(3) The sorted program/subroutines are compiled separately using the command file: LIP:MD (which was generated by TILSOS (see listing in item (10) below)). If an error were generated during compile, the compiler would indicate which subroutine had the error(s). This would not, however, inhibit the further completion of compilation.

(4) This is a listing of the "Object" files generated by the M7 @LIB command.

(5) Note that 33 blocks in six files were generated. These numbers are critical in that they are used to create a library in the next step.

(6) Using the "LIB" utility, a library "L1.OILB" is created. The decimal points are parts of the syntax in this command as the omission of them indicates octal numbers. The name
"L1" is used only because the "CUT2 L1" file, built by
TEXT is looking, arbitrarily, for a object library called
L1.
(7) Library L1.0LB is "stuffed." In the case of this
command file, each command is subsequently listed until
and <EOF> is encountered. In this example, six object
modules have been inserted into library L1.0LB.
(8) Using PIP, the library L1.0LB is renamed: OPTI.0LB.
This is the library name which will be used by OPTIM.00
when taskbuilding OPTIM.
(9) OPTIM is finally "taskbuilt."
(10) A listing of the files which have been generated
while building the executable module, OPTIM.0SM, and the
symbol table file, OPTIM.0TB. Note that the sum of the
space taken up by the files is over 1000 blocks.
(11) Housekeeping. Those files unnecessary to the execu-
tion of the OPTIM module are deleted by the seven PIP
directives shown.
(12) A listing of what remains: the two files necessary
to execute optimization.
LISTING OF BULLET, CHP

TIME

1. Ask does the two tapes mounted
2. IFQ-DOES GOTO 200
3. Ask does have you done a "PIE/PP"?
4. IFQ-DOES GOTO 200
5. Ask does there at least 6 blocks available
6. IFQ-DOES GOTO 200
7. Ask does this is going to take about 6 hours, ready
8. IFQ-DOES GOTO 200

FLX /PO=HT/ 20, 1 T1HLS.RCP.TSK/CO
FLX /PO=HT/ 20, 1 STUFF.TSK/DP
FLX * , * =T1HLS.RCP/CO
FLX * , * =STUFF.TSK/ CO
FLX * , TSK/DP
FLX /PO=HT/ 20, 1 * , * /DP
FLX /PO=HT/ 20, 1 * , * /DP
FLX /PO=HT/ 20, 1 HI2001, NHD/DP
FLX /PO=HT/ 20, 1 HI1202, V18/DP
FLX /PO=HT/ 20, 1 HI1203, V18/DP
FLX /PO=HT/ 20, 1 HI1304, NHD/DP
FLX /PO=HT/ 20, 1 HI1105, NHD/DP
FLX /PO=HT/ 20, 1 HI1205, V18/DP
FLX /PO=HT/ 20, 1 HI1306, V18/DP
FLX /PO=HT/ 20, 1 HI1106, V18/DP
FLX /PO=HT/ 20, 1 HI1206, V18/DP
FLX /PO=HT/ 20, 1 HI1307, V18/DP

THIS PAGE IS BEST QUALITY REPRODUCABLE
FROM COPY PUBLISHED NO 1982 ——
APPENDIX F

LISTING OF GLIDE TAPES
(IFS VERSION)

The following are listings of the contents of the tapes needed by BUILDGLD to build GIPMS. Though all the files could have fit on one tape, the method of dividing them was used to make the reading process more efficient. In general, the source listings are on MD0: (MD:) and the ED, CDL and existing GAIK files are on MD1:

The files were created under the MAK utility of MAK-111 in ICS format.
LISTING OF MTO:

DIRECTORY   MTO: [20, 1]
01-SEP-80

LIBR1.NEW  125.  01-SEP-80
LIBR2.NEW  172.  01-SEP-80
LIBR3.NEW  172.  01-SEP-80
LIBR4.NEW  260.  01-SEP-80
LIBR5.PUP  166.  01-SEP-80
BULKLB.NEW 351.  01-SEP-80
BULKM.NEW  577.  01-SEP-80
EDITM.NEW  449.  01-SEP-80
BULKF.NEW  20.   01-SEP-80
EDITLB.NEW 263.  01-SEP-80
STIFF.NEW  164.  01-SEP-80
DECOM.NEW  23.   01-SEP-80
STRESS.NEW 229.  01-SEP-80
AUTOL.NEW  26.   01-SEP-80
RESULT.NEW 544.  01-SEP-80
TRAN1.NEW  24.   01-SEP-80
TRAN2.NEW  26.   01-SEP-80
EECS.NEW   104.  01-SEP-80
LOCAL.NEW  120.  01-SEP-80
SAVER.NEW  8.   01-SEP-80
RESIDU.NEW 20.   01-SEP-80
PRINT.NEW  20.   01-SEP-80
TEST.NEW   2.    01-SEP-80
BULKS.NEW  646.  01-SEP-80
LOADS.NEW  551.  01-SEP-80
OPTIM.NEW  52.   01-SEP-80
DEFL.NEW   115.  01-SEP-80
TRANS.NEW  91.   01-SEP-80
DEFCS.NEW  152.  01-SEP-80
TEST2.NEW  3.    01-SEP-80
TSPTPL.NEW 6.    01-SEP-80
SUBS.NEW   52.   01-SEP-80

TOTAL OF 5333 BLOCKS IN 32 FILES
## Listing of MT1:

<table>
<thead>
<tr>
<th>Directory</th>
<th>MT1:(20+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILSR2.TSK</td>
<td>37. 01-SEP-80</td>
</tr>
<tr>
<td>EST.TSK</td>
<td>56. 01-SEP-80</td>
</tr>
<tr>
<td>STUFFE.TSK</td>
<td>41. 01-SEP-80</td>
</tr>
<tr>
<td>BUILD.CMD</td>
<td>9. 01-SEP-80</td>
</tr>
<tr>
<td>BUILDT.CMD</td>
<td>10. 01-SEP-80</td>
</tr>
<tr>
<td>EST.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>STIFF.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>SURG.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>TRAN1.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>TRANS.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>OPTIM.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>STRESS.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>EDITLB.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>SAVEK.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>PRINT.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>RESIDU.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>TRANS2.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>REDCS.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>GIFTSS.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>BULKM.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>BULKLB.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>DEFL.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>RESULT.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>BULKF.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>EDITM.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>AUTOL.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>DECOM.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>DEFC5.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>LOCAL.CMD</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>BUILDL.CMD</td>
<td>10. 01-SEP-80</td>
</tr>
<tr>
<td>DECOM.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>OPTIM.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>BULKM.ODL</td>
<td>3. 01-SEP-80</td>
</tr>
<tr>
<td>BULKLB.ODL</td>
<td>2. 01-SEP-80</td>
</tr>
<tr>
<td>DEFL.ODL</td>
<td>2. 01-SEP-80</td>
</tr>
<tr>
<td>EDITM.ODL</td>
<td>2. 01-SEP-80</td>
</tr>
<tr>
<td>LOCAL.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>SUBS.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>DEFC5.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>TRANS.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>STRESS.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>STIFF.ODL</td>
<td>2. 01-SEP-80</td>
</tr>
<tr>
<td>REDCS.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>RESULT.ODL</td>
<td>3. 01-SEP-80</td>
</tr>
<tr>
<td>EDITLB.ODL</td>
<td>1. 01-SEP-80</td>
</tr>
<tr>
<td>GIFTSS.INF</td>
<td>197. 01-SEP-80</td>
</tr>
<tr>
<td>EST.FTN</td>
<td>2. 01-SEP-80</td>
</tr>
<tr>
<td>FILSR2.FTN</td>
<td>21. 01-SEP-80</td>
</tr>
<tr>
<td>FILSR1.FTN</td>
<td>13. 01-SEP-80</td>
</tr>
<tr>
<td>STUFFE.FTN</td>
<td>6. 01-SEP-80</td>
</tr>
<tr>
<td>FILSR1.TSK</td>
<td>51. 01-SEP-80</td>
</tr>
</tbody>
</table>

Total of 520 blocks in 51 files.
BIBLIOGRAPHY


# INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Address</th>
</tr>
</thead>
</table>
| 1.  | 2      | Defense Technical Information Center  
Cameron Station  
Alexandria, Virginia 22314 |
| 2.  | 2      | Library, Code 0142  
Naval Postgraduate School  
Monterey, California 93940 |
| 3.  | 1      | Department Chairman, Code 69  
Department of Mechanical Engineering  
Naval Postgraduate School  
Monterey, California 93940 |
| 4.  | 5      | Professor Gilles Cantin, Code 6931  
Department of Mechanical Engineering  
Naval Postgraduate School  
Monterey, California 93940 |
| 5.  | 2      | Professor Hussein A. Hamel  
University of Arizona  
College of Engineering  
Aerospace and Mechanical Engineering Department  
Interactive Graphics Engineering Laboratory  
AEB Building, Room 210  
Tucson, Arizona 85721 |
| 6.  | 1      | ICDA John T. Sheldon, USN  
216 Camino Intrada  
Chula Vista, California 92010 |
| 7.  | 1      | Professor George A. Rahe, Code 521a  
Department of Computer Science  
Naval Postgraduate School  
Monterey, California 93940 |