DISCLAIMER

NRTSC
Technology
On Time
On Target

WSRC-RP-90-1129

KEYWORDS: ARCHY User Manual

RETENTION: PERMANENT

ARCHY USER'S MANUAL (U)

J. E. Aull
G. L. Hire
R. E. Pevey

ISSUED: NOVEMBER 1990

WESTINGHOUSE SAVANNAH RIVER COMPANY
SAVANNAH RIVER LABORATORY
AIKEN, SC 29808

PREPARED FOR THE U.S. DEPARTMENT OF ENERGY UNDER CONTRACT DE-AC09 88SR18035

DOCUMENT: WSRC-RP-90-1129

TITLE: ARCHY User's Manual (U)

TASK: 90-003-0

APPROVALS

M R Buckner DATE: 2-1-91

M. R. BUCKNER, MANAGER
SCIENTIFIC COMPUTATIONS SECTION

C. E. Apperson DATE: 2.4.91

C. E. APPERSON, MANAGER
REACTOR PHYSICS GROUP

SUMMARY

This document tells how to use the system of programs called ARCHY(Analysis and Reverse Engineering of Code with Hierarchy and Yourdon Diagrams). This document consists of an introduction which gives an overview of ARCHY and the problem that it solves, an explanation of how to use the ARCHY menu system, and detailed explanations of the menu choices within ARCHY. The structure of the ARCHY database is in Appendix A.

I. INTRODUCTION

ARCHY will be used to document and maintain existing programs and in the development of new programs. It runs under VAX/VMS Version 5 and is written in FORTRAN. When a program is read by ARCHY it creates a database, generates structure charts and data flow diagrams, restructures programs by automatically indenting, reorders statement labels, and maintains module headers.

There are two principle types of tasks that ARCHY is used for: development of new codes and analysis of existing codes. Each of these will require a different "flow" through the system.

A. Use of ARCHY as a Program Development Tool

A program development project using ARCHY would proceed in the following order:

(1) The computational goals would be conceptually laid out for the new program in a Problem Statement and Solution Proposal report (PSSP), which would include a description of the theory proposed for the solution.

(2) A conceptual design would proceed with the creation of Data Flow Diagrams (DFD) which would illustrate the flow of data from input files through transforms to output files. Based on

this information, the Transforms, Files, DFD, and Data Flow tables would be input into the database using the DB utility. The DATAFLOW utility would be used to document this specification in the PSSP.

(3) From this detailed data flow analysis, the program hierarchical structure would be developed, resulting in a correspondence between transforms and program modules. This correspondence would be put into the transforms structure using the DB utility; then the consistency checking in DB would guide the user to include the variables in the module lists. The STRUCT utility will then print the initial structure chart.

(4) Next, attention would turn to the individual modules of the program; Purpose, Algorithm, and Procedure descriptions would be included in the module structure of the database with the DB utility.

(5) The designer would then "flesh out" the variable list by deciding on the variable types, character lengths, array dimensions, and variable definitions; this would be included in the Data Dictionary table of the database by the DB utility.

(6) Implementation would begin by an initial run through SKELETON; this would create a complete skeleton of the system including all variables being passed through argument lists.

(7) When the designer is satisfied with the module descriptions and the interfaces between them (the subroutine calls and common blocks), design would be documented in a Program Design Report, including Module Design sheets (which ultimately become module headers) and a Design Sheet Data Dictionary (which contains all variable descriptions as well as a list of which modules use each variable) using the OUTPUT menu.

(9) Module Design sheets would be turned over to programming teams to code and test the modules. The ARCHY cleanup utilities (CLEAN, ALPHA, and LABEL) would be used in this development.

(10) As the modules are ready, they would be incorporated into a growing program, which would undergo system testing to provide verification and validation of the entire system. The coding and testing would be documented in a Program Implementation report.

(11) The completed full system would then undergo benchmark testing to provide validation of the new code; this would be documented in a Program Benchmark Testing Report.

B. Use of ARCHY as a Program Analysis Tool

An analysis of an existing FORTRAN code would proceed in a fashion similar to the following:

(1) The ARCHY cleanup utilities would be used to re-cast the FORTRAN coding into a consistent style.

(2) The DECIFE utility would operate on the FORTRAN coding and would create an ARCHY database.

(3) The DB utility completeness checks would be used to prompt the user for module descriptions and variable definitions.

(4) The INCORP utility would be used to put the headers into each module.

(5) The STRUCT and DESIGN utilities would be used to print the Structure Chart, Design Sheets, and Design Sheet Data Dictionary.

(6) The DB consistency checks would be used to aid the user in translating the modules into transforms and grouping the input and output variables into data flows.

(7) The DATAFLOW utility would be used to print a Data Flow Diagram, a DFD Data Dictionary, and a listing of the transform table for the program.

II. HOW TO USE THE ARCHY MENU SYSTEM

ARCHY runs on the Savannah River Laboratory VAXcluster. To run it, type RUN SRLUSER1:[T3895.ARCHY.SRC]ARCHY at the system prompt. That causes the ARCHY main menu to appear on your terminal as shown in Fig. 1. A brief explanation of what each menu choice does is shown within the menu. A detailed explanation of the menu choices is given in Section III. Each of the menu choices 1-8 cause a FORTRAN program to be executed. Choices 9, A, and B present sub-menus. Choice C lets you execute a DCL command while still in the ARCHY system. Choice D exits from the ARCHY system.

In Figure 1 what the user types is shown in **bold** type.

```
---------------------------- ARCHY MAIN MENU -------------
Input File: TEST

Output File:

1.  DECIFE        Create database from existing program.
2.  DB            Edit database using DB.
3.  SKELETON      Create skeleton program from database
4.  DHEADER       Transfer info. from header to database.
5.  INCORP        Insert database information into headers.
6.  MIX           Mix two databases.
7.  EXTRA         Extract subroutine from program.
8.  TRAMP         Fill in subprogram arguments in database.
9.  BEAUTIFY      Use cleanup utilities.
A.  OUTPUT        Print diagrams or reports.
B.  PREF          Set up preferences.
C.  DCL           Issue DCL command.
D.  EXIT          Return to operating system.

      Use the arrow keys or type number to select option.
```

Fig. 1 ARCHY Main Menu

You must type an input file name before choosing any of the menu options. The filename may consist of a maximum of 50 characters including node, device, directory, filename, extension, and version. You can leave off the extension and a default will be supplied. The default extension supplied by ARCHY depends on which of the menu options you choose. If you don't supply an output file name then ARCHY assumes that the output filename will be the same as the input filename though the extension may be different. The default input and output extensions for each of the ARCHY menu choices is listed in Table 1.

Table 1. Default Extensions

Menu	Choice	Input Extension	Output Extension
MAIN	1. DECIFE	FOR	RCH
MAIN	2. DB	RCH	RCH
MAIN	3. SKELETON	RCH	FOR
MAIN	4. DHEADER	FOR	RCH
MAIN	5. INCORP	RCH	FOR
MAIN	6. MIX	RCH	RCH
MAIN	7. EXTRA	FOR	FOR
MAIN	8. TRAMP	RCH	RCH
MAIN	9. BEAUTIFY	FOR	FOR
MAIN	A. OUTPUT	RCH	See Below
OUTPUT	1. STRUCT	RCH	SPS
OUTPUT	2. HEADER	RCH	MDS
OUTPUT	3. DATADICT	RCH	DDD
OUTPUT	4. DATAFLOW	RCH	FPS
OUTPUT	5. FLOWDICT	RCH	FDD
OUTPUT	6. TRANFORM	RCH	TDD

After entering the input filename(s) press RETURN and you see a highlight on the first menu choice. Move the highlight up or down with the arrow keys and press RETURN when the desired choice is highlighted. Another way to select a menu option is to type the number or letter that corresponds to that choice without pressing the RETURN key.

Option 9 presents the beautify menu which contains utilities for making programs more readable. If option 9 is chosen in Fig. 1 then you get the screen in Fig. 2.

```
------------------------------ ARCHY MAIN MENU -----------------------
Input File : TEST.FOR

Output File: TEST.FOR

------------------------- BEAUTIFY MENU --------------------------
|
|  1. CLEAN    Indent and resequence statement numbers.
|  2. ALPHA    Alphabetize modules.
|  3. LABEL    Add end-of-line labels.
|  4. PORT     Convert file for transmission to IBM.
|  5. PRETRN   Insert blank before each line.
|  6. PREV     Return to previous menu.
|
|     Use the UP/DOWN arrow keys to select option.
```

Fig. 2. Beautify Menu

Each of the beautify menu choices is described in detail in Section III.B.

Option A on the main menu presents the output menu which contains utilities for printing various diagrams and reports based on information in the ARCHY database. If option A is chosen in Fig. 1 then you get the screen in Fig. 3.

```
---------------------------- ARCHY MAIN MENU ----------------------------
Input File: TEST.RCH

Output File: TEST

------------------------ OUTPUT MENU -------------------------------
|
|              Print Queue: LASER5
|
|             1. STRUCT        Structure chart.
|             2. HEADER        Module Design Sheets
|             3. DATADICT      Design Sheet Data Dictionary
|             4. DATAFLOW      Dataflow Diagram
|             5. FLOWDICT      Dataflow Diagram Data Dictionary
|             6. TRANFORM      Transform Descriptions
|             7. PREV          Return to previous menu
|
|     Use the UP/DOWN arrow keys to select option.
```

Fig. 3. Output Menu

The print queue will be supplied from the preferences file. Notice that the choice of the output menu makes a default extension of "RCH" appear on the input file. Each of the choices 1-6 corresponds to a different default extension on the output file as shown in Table 1. Each of the output menu choices is described in detail in Section III.C.

Option B on the main menu activates the preferences menu which you can use to set parameters which control the execution of programs within the menu. When you choose this option you get a list of programs to choose from as shown in Fig. 4. The current list only includes three functions, ALPHA, CLEAN, and OUTPUT. Use the arrow keys to highlight your choice and press RETURN. You will then be asked to answer questions that indicate your preference for that function. Your answers are stored in a file called ARCHY.PREF in your root directory. Then when you execute the program in question, the preference file is read by that program.

```
---------------------------- ARCHY MAIN MENU ----------------------------
Input File: TEST

Output File:

------------------ PREFERENCES MENU --------------------------
|
|       What function do you want to set preferences for ?
|
|  ALPHA      CLEAN      OUTPUT
|
|
|
|
```

Fig. 4. Preferences Menu

The questions that correspond to each of the functions shown in Fig. 4 are shown in Table 2. ALPHA alphabetizes modules within a program and you are given a choice as to whether you want the main module at the beginning of the program or in its alphabetic slot. CLEAN can put a frame of stars around comment lines if you desire and FORMAT statements will be located at the end of modules if you so choose. The print queue where reports from the output menu are sent is entered in the OUTPUT function of the preferences menu. The default answer to each of the yes or no questions in Table 2 is "yes". The default output queue is LASER5.

Table 2. Preference Questions

ALPHA
Locate main module first?(Y/N)
CLEAN
Put frame around comment lines?(Y/N)
Put FORMAT statements at end of modules?(Y/N)
OUTPUT
Print queue :

When you return to the main menu after running a program you see the last entry typed in the input field. Press return at this point to reuse that filename. Typing any other key invalidates what you see in that field and causes the input filename to be what you type, even if the only thing you type is a space.

III. Program Documentation

This section contains detailed instructions for running each of the programs within the ARCHY menu system. Subsection A deals with programs accessed through the main menu, subsection B covers the beautify menu, and subsection C contains information about the output menu.

A. Main Menu

1. DECIFE: Create Database From Existing Program

DECIFE reads a FORTRAN program and outputs an ARCHY database. Figure 1.1 is an example of a program and the resulting data base output is in Figure 1.2.

```
      PROGRAM TEST1
C     A SIMPLE EXAMPLE
      REAL*8 X,XBAR
      X=2.
      Y=X**3.
      CALL CROOT(Z,Y)
      V=XHALF(Z)
      WRITE(5,100)V
100   FORMAT(1X,'V= ',F10.3)
      STOP
      END
      SUBROUTINE CROOT(S,T)
      S=T**.333
      RETURN
      END
      FUNCTION XHALF(X)
      XHALF=X/2.
      RETURN
      END
```

Fig. 1.1. Program Fed Into DECIFE

Figure 1.2 shows the ARCHY database file, TEST1.RCH, that is output when TEST1.FOR is analyzed by DECIFE. Note that the variable X is declared as REAL*8 in the main program, TEST1, yet X is implicitly typed as REAL*4 in the function XHALF, yet only one type is shown in the Data Dictionary table below. Refer to Appendix A for an explanation of the database structure.

```
#Modules<32>
@test1
:Structure chart description<16>
:Purpose<64>
:History<64>
:Called modules<32>[Modules]
croot
xhalf
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
```

```
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
v
x
y
z
:Theory<64>
:Algorithm<64>
@croot
:Structure chart description<16>
:Purpose<64>
:History<64>
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
t
:External variables - set<32>[Data Dictionary]
s
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
:Theory<64>
:Algorithm<64>
@xhalf
:Structure chart description<16>
:Purpose<64>
:History<64>
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
x
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
xhalf
:Theory<64>
:Algorithm<64>
#Data Dictionary<32>
@
:Type<16>
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@s
:Type<16>
r*4
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@t
```

```
:Type<16>
r*4
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@v
:Type<16>
r*4
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@x
:Type<16>
R*8
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@xhalf
:Type<16>
r*4
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@y
:Type<16>
r*4
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@z
:Type<16>
r*4
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
#COMMON blocks<32>
@
:Description<64>
:Modules using<32>[Modules]
:Variable names<32>[Data Dictionary]
#Transforms<32>
@
:Transform number<16>
:Transform title<32>
:Description<64>
```

```
:Procedure<64>
:Incoming data flows<64>[Data flows]
:Outgoing data flows<64>[Data flows]
:Subordinate transforms<32>[Transforms]
:Corresponding modules<32>[Modules]
#Files<64>
@
:File number<16>
:File title<32>
:Description<64>
:Incoming data flows<64>[Data flows]
:Outgoing data flows<64>[Data flows]
#Data flows<64>
@
:Description<64>
:Variable names<32>[Data Dictionary]
#DFD<64>
@
:DFD Title<64>
:Description<64>
:Displayed transforms(x,y)<32>[Transforms]
:Displayed files(x,y)<64>[Files]
:Extra incoming dataflows<64>[Data flows]
:Extra outgoing dataflows<64>[Data flows]
:Legend position<32>
```

Fig. 1.2. Database Output by DECIFE

2. DB: Edit Database Using DB

DB is a database editor that allows you to browse, create, edit, check, and print information contained in an ARCHY database file. An ARCHY database file is actually just an ASCII file so it can be easily edited with any editor. DB provides the functionality of a full screen editor and presents ARCHY data in an easy-to-use format along with markers for field widths.

a. Using the DB Menu

The main menu of the database manager lets you choose which table you want to edit. In Fig. 2.1 a "1" was chosen which means work with the MODULES table.

```
|---------------------- Working with dtest1.rch ----------------------------
||
||
||                        Desired Table ==> 1
||
||                         1. Modules
||                         2. Data Dictionary
||                         3. COMMON blocks
||                         4. Transforms
||                         5. Files
||                         6. Data flows
||                         7. DFD
||                         c. Check the database
||                         q. QUIT without saving
||                         s. SAVE results without exiting
||                         x. EXIT with save
```

Fig. 2.1 DB Main Menu

The check option is very useful for performing consistency checks on the database which could point out problems within the program from which the database is generated. Figure 2.2 shows the menu you get when the check option is chosen by typing "C" on the main DB menu screen.

```
 1. Check that all referenced entries exist
 2. Check that all variables referenced
 3.    "    "    "  data flows     "
 4.    "    "    "  transforms     "
 5.    "    "    "   modules       "
 6.    "    "    "    files        "
 7. Check that Transforms and subordinates consistent
 8. Check that Modules and subroutines consistent
 9. Check that Module/Transforms consistent
```

Fig. 2.2 Consistency Check Menu

Table 2.1 shows some of the messages generated when inconsistencies are found.

Table 2.1 Consistency Check Error Messages

Module _____ needs subordinate variable _____
The following referenced entries do not exist:
Table _____ needs entry _____
The following _____ entries are not referenced:'
Transform _____ subs need incoming
Transform _____ subs need outgoing _____
Transform _____ needs incoming _____
Transform _____ needs outgoing _____

On the edit screen, shown in Fig. 2.3, you choose between the various choices that appear on the top line. You can choose by moving the cursor from word to word using arrow keys or by typing the first letter in the word and then pressing RETURN. If you choose any of the first four (browse, edit, delete, or rename) then you will have to supply an entry name. Entering "all" at the "Choose an entry ==>" prompt will allow you to cycle through all the current entries in order. If you just press RETURN at the "Choose an entry ==>" prompt, then you can use the arrow keys to highlight the entry desired from the "Current entries" list.

Notice that all the names for the current entries are shown on the screen. If you want to add an entry then choose "EDIT" and type the name of the new entry. Then it will ask if you want to add it. Alphabetize just sorts the entries on the screen. CHECK does a consistency check with other tables in the database. In Figure 2.2 an edit of the CROOT entry in the MODULES table was chosen.

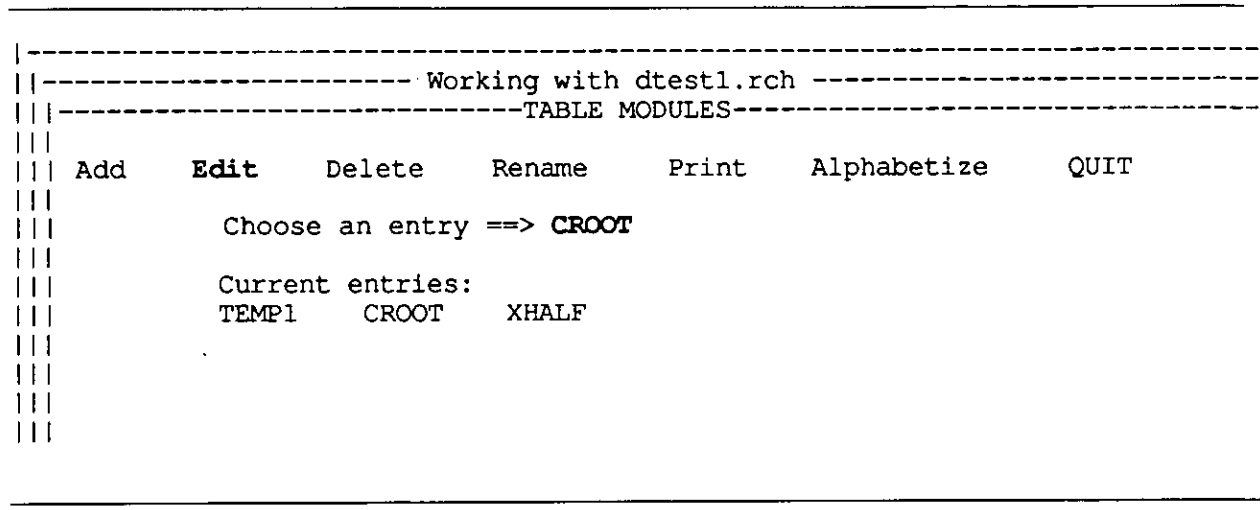


Fig. 2.3 DB Table Menu

Figure 2.4 shows how the CROOT record in the MODULES table is edited. You just arrow down to the field you want to change, press RETURN, and type in what you want. The field lengths are marked with "|" characters. When you are done with that record press the PF2 key to get back to the previous screen.

```
|-------------------------------------------------------------------------
||---------------------- Working with dtest1.rch --------------------------
|||-----------------------------TABLE MODULES-------------------------------
||||--------------PRESS PF2 TO STOP EDITING ENTRY: CROOT---------------------
|||||-------------------------------------------------------------------------
|||||Structure chart description
|||||
|||||Purpose
|||||
|||||History
|||||
|||||Called modules
|||||
|||||Parameters(value)
|||||
|||||External variables - used
|||||
|||||External variables - set
|||||
|||||External variables - just passing through
|||||
|||||Internal variables
|||||
|||||Theory
|||||
|||||Algorithm
|||||
|||||-------------------------------------------------------------------------
```

Fig. 2.4 DB Edit Screen

All the tables are edited the same way. The only difference is in the names of the fields within the different tables. Once an entry has been selected for edit, the you will see a new panel containing the current contents of the entry. The categories will be printed left-justified and highlighted. **These cannot be changed by the editor.** The current values (if any) for each category will be listed below the category titles. They are indented three characters and printed in normal (non-highlighted) intensity. These can be modified by the editor. Between the list of values and the next

category title is a blank line (there for aesthetics) **which cannot be written on.**

b. Using the DB Editor

1) Getting out of the editor

When you want out use one of the following three options.

a) Press PF2 or F2 which is the equivalent of the COMMAND line command EXIT. (See COMMAND line commands below).

b) Use the COMMAND line command EXIT - which exits **with save.**

c) Use the COMMAND line command QUIT - which exits **without save** (and also jumps you out of the automatic scrolling through entries begun by specifying entry "all").

2) Positioning the cursor

Editing takes place at the position of the cursor. The cursor can be moved using:

a) The arrow keys;

b) The Number Keypad (with NUMLOCK on):
   1 = end of the line (plus one character)
   2 = down 1 line
   3 = down 1 page
   4 = left 1 character
   5 = no action
   6 = right 1 character
   7 = first character of line
   8 = up 1 line
   9 = up 1 page

c) The TAB key - takes cursor to first line of next category;

d) COMMAND line commands TOP, BOT, FIND. (See "COMMAND line commands" below.)

3) Adding a line under a category

To add a line to the end of a category, position the cursor in the blank line at the end of the list and press "Enter". This will create a new field at the bottom of the list (with the cursor positioned at the first character of the field); this field can then be edited as in "Editing an existing line" below.

4) Editing an existing line

To edit an existing line, position the cursor in the line to be edited. (Remember, the blank line at the end of a list of values **cannot be edited**. If you want to write there, first press "Enter" to create a field.)

When a line has been chosen, the **margin markers** (broken vertical lines) will be visible. These mark the edges of the field as it will be saved. Two items of note:

a) It is possible to write beyond the right edge of the margin, but anything written will not be saved in the database.

b) If a longer line is present, the right margin marker is not part of the line. There is a character hidden under it.

With the cursor positioned where you want it, the following special keys are active:

a) The "0" on the KEYPAD toggles between INSERT and OVERSTRIKE mode;

b) The period on the KEYPAD deletes the character at the cursor (and collapses the line in INSERT mode);

c) The BACKSPACE or DELETE key deletes the previous character and collapses the line;

d) The F4 or PF4 key erases to the end of the line;

e) The ENTER key creates a new line and moves the rest of the current line there;

5) COMMAND line commands

Extra user power is provided when PF1 or F1 is pressed. This initiates a COMMAND line which allows the following commands (in either upper or lower case):

BOT = Moves cursor to bottom of entry
TOP = Moves cursor to top of entry
FIND = Prompts user for a string; the cursor moves to the next occurrence of the string (Same as PF3)
ALP = Alphabetizes the lines of the current category
LEN = Arranges the lines in order of length in the current category (shortest to longest)
QUIT = Leaves the editor **without save**
EXIT = Leaves the editor **with save** (Same as PF2)

6) PF Keys

PF1 - Invokes COMMAND line (See previous paragraph)
PF2 - Leaves the editor with save (Same as COMMAND line command 'EXIT')

PF3 - Prompts user for a string; the cursor moves to the next occurrence of the string (Same as COMMAND line command 'FIND')
PF4 - Erases to end of line

3. SKELETON: Create Skeleton Program From Database

The SKELETON program is part of the ARCHY system. SKELETON creates a compilable FORTRAN program from a specified ARCHY database.

a. What SKELETON Does

SKELETON produces a FORTRAN output file from the input ARCHY database. The output file includes the call and subroutine statements with their appropriate passed arguments. SKELETON also creates the parameter statements with their appropriate values and produces the type statements for all the variables. SKELETON uses the Modules and Data Dictionary tables of the ARCHY input database to produce the FORTRAN output file. The Data Dictionary table is cross referenced to include the variable type and definition. Figure 3.1 shows the Tables and categories that SKELETON uses to produce the FORTRAN file. The database tables are defined by a "#", table entries are defined by an "@", and the categories are distinguished by a ":".

b. How To Use SKELETON

1). Enter the input and output files on the ARCHY main menu screen. The default output file is a FORTRAN file with the same name as the input file. Figure 3.2 shows a sample execution of SKELETON. In this figure what the user types is enclosed in braces.
2). Choose SKELETON from the main menu. The default extension when SKELETON is chosen from the main menu is ".RCH".

After you choose SKELETON from the main menu, you will be asked for the "Name of the top module". Figure 3.3 shows the input ARCHY database and Figure 3.4 shows the FORTRAN output file after the execution of SKELETON.

```
#Modules<32>
@
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
#Data Dictionary<32>
@
:Type<16>
:Array limits<32>
```

Figure 3.1. Database Tables And Categories Used By SKELETON

```
------------------------- ARCHY MAIN MENU -----------------------------
INPUT FILE:  main.RCH
OUTPUT FILE:  MAIN.FOR
----------------------------- SKELETON ---------------------------------
  |
  |
  |   Name of the top module ==>
  |  {main}
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
  |
---------------------------------------------------------------------------
```

Figure 3.2. Sample Execution Of SKELETON

```
#Modules<32>
@main
:Called modules<32>[Modules]
a
b
c
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
one
three
two
@a
```

```
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
one
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
@b
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
two
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
@c
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
three
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
#Data Dictionary<32>
@one
:Type<16>
C*7
:Array limits<32>
@three
:Type<16>
C*11
:Array limits<32>
@two
:Type<16>
C*7
:Array limits<32>
```

Figure 3.3. Input ARCHY Database

```
      program main
c     PARAMETERS
c     EXTERNAL variables
c     INTERNAL variables
      character*7 one
      character*11 three
      character*7 two
      write(*,9010)
 9010 format( 2x,' main                          ')
      call a(one)
      call b(two)
      call c(three)
      stop
      end
c*************************************************************************
      subroutine a(one)
c     PARAMETERS
c     EXTERNAL variables
      character*7 one
c     INTERNAL variables
      write(*,9010)
 9010 format( 4x,' a                             ')
      return
      end
c*************************************************************************
      subroutine b(two)
c     PARAMETERS
c     EXTERNAL variables
      character*7 two
c     INTERNAL variables
      write(*,9010)
 9010 format( 4x,' b                             ')
      return
      end
c*************************************************************************
      subroutine c(three)
c     PARAMETERS
c     EXTERNAL variables
      character*11 three
c     INTERNAL variables
      write(*,9010)
 9010 format( 4x,' c                             ')
      return
      end
```

Figure 3.4. Output File After Execution Of SKELETON

c. Requirements For The Operation Of SKELETON

This section describes some limitations and useful information which may aid you in your operation of SKELETON.

1) Limitations

SKELETON will only allow the database to have a maximum of 100,000 lines 80 characters long. SKELETON may allocate up to 24.5 Megabytes of virtual memory.

2) Information

SKELETON requires the user to input the name of the top module exactly as it appears in the Modules table of the ARCHY database with respect to upper or lower case. The created FORTRAN file contains write statements which print out the names of the modules when the created file is executed.

4. DHEADER: Transfer Info From Header to Database

The DHEADER program is part of the ARCHY system. DHEADER decifers subroutine headers and inserts the descriptive information that it gleans from comment statements in the header into an ARCHY database.

a. What DHEADER Does

DHEADER reads a FORTRAN program. At the beginning of each module it looks for the header block which is enclosed in stars. Then it looks for specific markers that occur within each ARCHY header. The markers that pertain to the module are shown in Table 4.1.

DHEADER finds the descriptive text that occurs after each marker and inserts that text into the corresponding field within the Modules table in the ARCHY database. DHEADER also reads variable definitions from program headers and inserts them into the Data Dictionary table.

Table 4.1 Markers Recognized by DHEADER

Module name:
Structure chart description:
Purpose:
History:
Theory:
Algorithm:

b. How To Use DHEADER

Figure 4.1 shows how to execute DHEADER. What you type is shown inside {braces}. The extensions supplied by default are shown in **bold** type. Simply type the name of the program whose headers are being decifered at the "Input File:" prompt and the name of the database being created at the "Output File:" prompt. Then choose option 4 from the main menu. ARCHY then prompts you for the name of the input database. At this point you type the name of the database which will have information from the header inserted into it. When you press RETURN, ARCHY echoes the three file names and the information is transferred from the header to the database.

The program that is the source of the descriptive information in this example is shown in Fig. 4.2. The databases before and after execution of DHEADER are shown in Figures 4.3 and 4.4 respectively.

```
---------------------------- ARCHY MAIN MENU -----------------------
Input File : {dhtest2in.FOR}

Output File: {dhtest2out.RCH}

---------------------SPECIAL INPUT FOR DHEADR-----------------------
|
|      Name of input database: {dhtest2in.RCH}
|
|
|
|
|
```

Fig. 4.1 Sample Execution of DHEADER

```
c*************************************************************************#
c                                                                         #
c Module name:                                                            #
c   dhtest2                                                               #
c                                                                         #
c Structure chart description:                                            #
c   EXTRACTS CODE                                                         #
c   FROM AN EXISTING                                                      #
c   FORTRAN FILE                                                          #
c                                                                         #
c Purpose:                                                                #
c   TO EXTRACT CODING FOR MODULES FROM A FORTRAN FILE AND EITHER          #
c   APPEND IT TO ANOTHER FORTRAN FILE OR CREATE A NEW FORTRAN FILE.       #
c                                                                         #
c Called modules:                                                         #
c   NONE                                                                  #
c                                                                         #
c Internal variables:                                                     #
c   A80                                                          C*80     #
c         A line of ALINE.                                                #
c   ALINE (50000)                                                C*80     #
c         An array containing an entire module                            #
c   I                                                             I*4     #
c         Counting variable                                               #
c*************************************************************************#
      character*80 a80
      character*80 aline(50000)
      do i=1,189
       read(*,100)a80
       aline(i)=a80
      enddo
      stop
100   format(a)
```

Fig. 4.2 Program Input Into DHEADER

```
#Modules<32>
@dhtest2
:Structure chart description<16>
:Purpose<64>
:History<64>
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
a80
aline
i
:Theory<64>
:Algorithm<64>
#Data Dictionary<32>
@
:Type<16>
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@a80
:Type<16>
C*80
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@aline
:Type<16>
C*80
:Array limits<32>
50000
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@i
:Type<16>
i*4
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
```

Fig. 4.3 Database Before DHEADER Adds Descriptions

```
#Modules<32>
@dhtest2
:Structure chart description<16>
 EXTRACTS CODE
 FROM AN EXISTING
 FORTRAN FILE
:Purpose<64>
 TO EXTRACT CODING FOR MODULES FROM A FORTRAN FILE AND EITHER
 APPEND IT TO ANOTHER FORTRAN FILE OR CREATE A NEW FORTRAN FILE.
:History<64>
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
a80
aline
i
:Theory<64>
:Algorithm<64>
#Data Dictionary<32>
@
:Type<16>
:Array limits<32>
:Definition<64>
:Modules using<32>[Modules]
:Aliases<64>[Data Dictionary]
@a80
:Type<16>
C*80
:Array limits<32>
:Definition<64>
    A line of ALINE.
@aline
:Type<16>
C*80
:Array limits<32>
50000
:Definition<64>
    An array containing an entire module
@i
:Type<16>
i*4
:Array limits<32>
:Definition<64>
    Counting variable
```

Fig. 4.4 Database After DHEADER Adds Descriptions

c. DHEADER Limitations and Requirements

DHEADER reads the entire program in one pass. As it encounters markers and variable names, it stores them in two temporary files, TEMPM.TMP and TEMPD.TMP. It then merges each of these files with the input database. If a description already exists in the input database, it will be overwritten by the description from the program header. It is possible that a program could have a variable that has the identical name, type, and dimension within two different modules yet the definitions of those two variables could be different. In a case like this, DHEADER, will insert the definition that it first encounters within the TEMPD.TMP file. The first one that it encounters might not coincide with the first occurrence of that variable definition within a module header because DHEADER begins a sequential search of the TEMPD.TMP file where the last search left off and then TEMPD.TMP is rewound and the search resumes at the top of the file. It is very important to have the header in the exact ARCHY format in order for DHEADER to extract the descriptive information. A few of the specifications for the ARCHY header format are summarized in the following list.

1) The header must be surrounded by a "frame" of stars with the pound sign in column 65.

2) The markers must match the ones listed in Table 4.1 exactly to the letter.

3) The blank lines within the header frame are essential and extra blank lines can be deadly.

4) Variable names, types, and dimensions must match the database exactly (except for case) with variable names beginning in column 5, dimensions within parentheses following the variable name, and variable types ending in column 63.

5) The module used to match within the database comes from the header rather than from within a SUBROUTINE or FUNCTION statement.

5. INCORP: Insert Database Information Into Headers

The INCORP program is part of the ARCHY system. INCORP creates subroutine headers and inserts them at the beginning of the appropriate subroutine in a FORTRAN file.

a. What INCORP Does

INCORP produces a new version of the input file. The output file contains the original file with subroutine headers added at the start of each subroutine. INCORP reads a FORTRAN program and an ARCHY database. The Modules and Data Dictionary tables of the database are used to produce the subroutine headers which contain the categories from the Modules table. The Data Dictionary table is cross referenced to include the variable type and definition. Figure 5.1 shows the Tables and categories that INCORP uses to produce the headers. The database tables are defined by a "#", table entries are defined by an "@", and the categories are distinguished by a ":".

b. How To Use INCORP

1) Enter the input and output files on the ARCHY main menu screen. The default for the output file is a new version of the FORTRAN input file. Figure 5.2 shows a sample execution of INCORP.

2) Choose INCORP from the main menu. The default extension when INCORP is chosen from the main menu is ".FOR".

After you choose INCORP from the main menu, you will be asked for the "Input ARCHY database". The default extension is ".RCH". As INCORP is executed a list of the subroutine headers created by INCORP will appear on the screen. Figure 5.3 shows the FORTRAN input file and Figure 5.4 shows

the input ARCHY database. Figure 5.5 shows the FORTRAN output file after the execution of INCORP.

```
#Modules<32>
@
:Structure chart description<16>
:Purpose<64>
:History<64>
:Called modules<32>[Modules]
:Parameters(value)<64>
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
:Theory<64>
:Algorithm<64>
#Data Dictionary<32>
@
:Type<16>
:Definition<64>
```

Figure 5.1. Database Tables And Categories Used By INCORP

```
------------------------- ARCHY MAIN MENU -----------------------------
INPUT FILE:  main.FOR
OUTPUT FILE:  MAIN.FOR
------------------------------ INCORP ---------------------------------
   |
   |
   |   Input ARCHY database ==>
   | {main.RCH}
   |
   |  a
   |  b
   |  c
   |
   |
   |
```

Figure 5.2. Sample Execution Of INCORP

```
      program main
      character*7 one,two
      character*11 three
      one='This is'
      two=' a test'
      three=' of INCORP.'
      call a(one)
      call b(two)
      call c(three)
      stop
      end
      subroutine a(one)
      character*7 one
      write(*,10)one
10    format(1x,a7)
      return
      end
      subroutine b(two)
      character*7 two
      write(*,20)two
20    format(a7)
      return
      end
      subroutine c(three)
      character*11 three
      write(*,30)three
30    format(a11)
      return
      end
```

Figure 5.3. FORTRAN Input File

```
#Modules<32>
@main
:Structure chart description<16>
The main program
:Purpose<64>
To call a, b, and c.
:Subroutines called<32>[Modules]
a
b
c
:Internal variables<32>[Data Dictionary]
one
three
two
@a
:Structure chart description<16>
Writes one
:Purpose<64>
To write 'This is'.
:History<64>
Called by main.
:External variables - used<32>[Data Dictionary]
one
@b
:Structure chart description<16>
Writes two
:Purpose<64>
To write ' a test'.
:History<64>
Called by main.
:External variables - used<32>[Data Dictionary]
two
@c
:Structure chart description<16>
Writes three
:Purpose<64>
To write ' of INCORP.'.
:History<64>
Called by main.
:External variables - used<32>[Data Dictionary]
three
#Data Dictionary<32>
@one
:Type<16>
C*7
:Definition<64>
Contains the first two words of the sentence.
@three
```

```
:Type<16>
C*11
:Definition<64>
Contains the last two words of the sentence.
@two
:Type<16>
C*7
:Definition<64>
Contains the third and forth words of the sentence.
```

Figure 5.4. Input ARCHY Database

```
      program main
      character*7 one,two
c*************************************************************************#
c                                                                         #
c Module name:                                                            #
c   MAIN                                                                  #
c                                                                         #
c Structure chart description:                                            #
c   THE MAIN PROGRAM                                                      #
c                                                                         #
c Purpose:                                                                #
c   TO CALL A, B, AND C.                                                  #
c                                                                         #
c Subroutines called:                                                     #
c   A                                                                     #
c   B                                                                     #
c   C                                                                     #
c                                                                         #
c Internal variables:                                                     #
c   ONE                                                          C*7      #
c         Contains the first two words of the sentence.                   #
c   THREE                                                        C*11     #
c         Contains the last two words of the sentence.                    #
c   TWO                                                          C*7      #
c         Contains the third and forth words of the sentence.             #
c                                                                         #
c*************************************************************************#
      character*11 three
      one='This is'
      two=' a test'
      three=' of INCORP.'
      call a(one)
      call b(two)
      call c(three)
      stop
```

```
      end
      subroutine a(one)
      character*7 one
c*************************************************************************#
c                                                                         #
c Module name:                                                            #
c   A                                                                     #
c                                                                         #
c Structure chart description:                                            #
c   WRITES ONE                                                            #
c                                                                         #
c Purpose:                                                                #
c   TO WRITE 'THIS IS'.                                                   #
c                                                                         #
c History:                                                                #
c   CALLED BY MAIN.                                                       #
c                                                                         #
c External variables - used:                                              #
c   ONE                                                          C*7      #
c         Contains the first two words of the sentence.                   #
c                                                                         #
c*************************************************************************#
      write(*,10)one
10    format(1x,a7)
      return
      end
      subroutine b(two)
      character*7 two
c*************************************************************************#
c                                                                         #
c Module name:                                                            #
c   B                                                                     #
c                                                                         #
c Structure chart description:                                            #
c   WRITES TWO                                                            #
c                                                                         #
c Purpose:                                                                #
c   TO WRITE ' A TEST'.                                                   #
c                                                                         #
c History:                                                                #
c   CALLED BY MAIN.                                                       #
c                                                                         #
c External variables - used:                                              #
c   TWO                                                          C*7      #
c         Contains the third and forth words of the sentence.             #
c                                                                         #
c*************************************************************************#
      write(*,20)two
20    format(a7)
      return
      end
```

```
      subroutine c(three)
      character*11 three
c*************************************************************************#
c                                                                         #
c Module name:                                                            #
c   C                                                                     #
c                                                                         #
c Structure chart description:                                            #
c   WRITES THREE                                                          #
c                                                                         #
c Purpose:                                                                #
c   TO WRITE ' OF INCORP.'.                                               #
c                                                                         #
c History:                                                                #
c   CALLED BY MAIN.                                                       #
c                                                                         #
c External variables - used:                                              #
c   THREE                                                          C*11   #
c         Contains the last two words of the sentence.                    #
c                                                                         #
c*************************************************************************#
      write(*,30)three
30    format(a11)
      return
      end
```

Figure 5.5. Output File After Execution Of INCORP

c. Requirements For The Operation Of INCORP

This section describes some limitations and useful information which may aid you in your operation of INCORP.

1) Limitations

INCORP will only allow the databases to have a maximum of 50,000 lines 80 characters long. INCORP may allocate up to 8 Megabytes of virtual memory.

2) Information

INCORP allows the user to enter their responses in upper or lower case. The module names within the FORTRAN file must match the

modules in the ARCHY database exactly with regard to upper and lower case. The subroutine header will only contain the categories that have a description in the database, however the Module name will always be in the header. INCORP will not include a variable in the header if the variable is not listed in the Data Dictionary table of the ARCHY input file. The category descriptions, except for the variable definitions are written in upper case in the subroutine header.

6. MIX: Mix Two Databases.

The MIX program is part of the ARCHY system. MIX merges two ARCHY databases together to form one combined database.

a. What MIX Does

MIX reads two ARCHY databases and merges them together into one new database. MIX adds descriptions to certain categories in the Modules, Data Dictionary and the COMMON blocks tables of the input database, provided the category doesn't already contain a description. Figure 6.1 shows the Tables and categories that descriptions are added to by MIX. The tables are defined by a "#", table entries are defined by an "@", and the categories are distinguished by a ":". The Transforms, Files, Data flows and the DFD tables in the database are either left untouched or totally replaced.

b. How To Use MIX

1. Enter the input file (the name of the database you want to make additions to) on the ARCHY main menu screen. Enter the output file on the ARCHY Main Menu screen. The default output file is a new version of the input file. Figure 6.2 shows the execution of MIX for two different databases. The user responses are shown inside the braces, { }.

2. Choose MIX from the main menu. ".RCH" is the default extension when MIX is chosen from the main menu.

After you choose MIX from the main menu, you will be asked for the "Name of the database containing the descriptions". The default is the previous version of the input file. Figure 6.3 shows an execution of MIX where the default database is used. You will also be asked "Do you want the TRANSFORMS, FILES, DATAFLOWS, and DFD tables replaced in the database [YES]?". The default to this question is "YES". The answer to this question can be "yes" or "no" or just the first letter of the word and can be in either upper or lower case. If you enter "YES" or default your answer the TRANSFORMS, FILES, DATAFLOWS, and DFD tables of the input file are replaced by the tables in the database containing the descriptions. If you enter "n" or "no" the TRANSFORMS, FILES, DATAFLOWS, and DFD tables of the input file remain unchanged. Figure 6.4 shows the input file and Figure 6.5 shows the database with the descriptions prior to the execution of MIX, and Figure 6.6 shows the merged database after the execution of MIX.

```
#Modules<32>
:Structure chart description<16>
:Purpose<64>
:History<64>
:Theory<64>
:Algorithm<64>
#Data Dictionary<32>
:Definition<64>
#COMMON blocks<32>
:Description<64>
```

Figure 6.1. Database Tables And Categories Affected By MIX

```
------------------------- ARCHY MAIN MENU -----------------------------
INPUT FILE:  main.RCH
OUTPUT FILE:  MAIN.RCH
  ------------------------------ MIX ----------------------------------
  |
  |
  |   Name of database containing the descriptions:
  | {descript.RCH}
  |
  |   Do you want the TRANSFORMS, FILES, DATAFLOWS, and the DFD tables
  |   replaced in the database [YES]?
  |  {n}
  |
  |
```

Figure 6.2. Sample MIX Execution Of Two Different Databases

```
------------------------- ARCHY MAIN MENU -----------------------------
INPUT FILE:  main.rch;6
OUTPUT FILE:  main.rch;7
  ------------------------------ MIX ----------------------------------
  |
  |
  |   Name of database containing the descriptions:
  |  main.rch;5
  |
  |   Do you want the TRANSFORMS, FILES, DATAFLOWS, and the DFD tables
  |   replaced in the database [YES]?
  |  {n}
  |
  |
  |
  |
  |
  |
  |
  |
  |
```

Figure 6.3. Sample MIX Execution By Default

```
#Modules<32>
@mix
:Structure chart description<16>
Merge Two
Databases Into
One
:Purpose<64>
To combine a database containing descriptions with an empty database.
:History<64>
:Algorithm<64>
@search
:Structure chart description<16>
:Purpose<64>
:History<64>
:Algorithm<64>
#Data Dictionary<32>
@category
:Definition<64>
@entry
:Definition<64>
@table
:Definition<64>
```

Figure 6.4. Input File Prior To The Execution Of MIX

```
#Modules<32>
@mix
:Structure chart description<16>
:Purpose<64>
:History<64>
The main coding of the program.
:Algorithm<64>
For each category that mix deals with:
  Find the matching spot in the database containing the descriptions and
  write the description to the output file if it has one.
@search
:Structure chart description<16>
Finds The
Desired Text
:Purpose<64>
To look through a database and find the proper description for a
given table, entry and category.
:History<64>
Called by mix
:Algorithm<64>
Loop over the lines of the database containing the descriptions, until
the table, entry and category match and then write the description to
the output file if a description is found.
#Data Dictionary<32>
@category
:Definition<64>
The name of the category in the database.
@entry
:Definition<64>
The name of the entry in the database.
@table
:Definition<64>
The name of the table in the database.
```

Figure 6.5. Database Containing The Descriptions

```
#Modules<32>
@mix
:Structure chart description<16>
Merge Two
Databases Into
One
:Purpose<64>
To combine a database containing descriptions with an empty database.
:History<64>
The main coding of the program.
:Algorithm<64>
For each category that mix deals with:
  Find the matching spot in the database containing the descriptions and
  write the description to the output file if it has one.
@search
:Structure chart description<16>
Finds The
Desired Text
:Purpose<64>
To look through a database and find the proper description for a
given table, entry and category.
:History<64>
Called by mix
:Algorithm<64>
Loop over the lines of the database with descriptions, until the
table, entry and category match and then write the description to the
output file if a description is found.
#Data Dictionary<32>
@category
:Definition<64>
The name of the category in the database.
:Modules using<32>[Modules]
@entry
:Definition<64>
The name of the entry in the database.
:Modules using<32>[Modules]
@table
:Definition<64>
The name of the table in the database.
```

Figure 6.6. Output File After Execution Of MIX

c. Requirements For The Operation Of MIX

This section describes some limitations and useful information which may aid you in your operation of MIX.

1) Limitations

MIX will only allow the databases to have a maximum of 50,000 lines 80 characters long. MIX may allocate up to 8 Megabytes of virtual memory.

2) Information

MIX allows the user to enter their responses in upper or lower case.

7. EXTRA: Extract Subroutine From Program

The EXTRA program is part of the ARCHY system. EXTRA extracts modules from an existing FORTRAN program and either creates a new FORTRAN file or appends the modules to an existing FORTRAN program.

a. What EXTRA Does

EXTRA extracts coding from a FORTRAN program and creates a new FORTRAN file containing the specified subroutines or appends the subroutines to an existing FORTRAN program. EXTRA uses the Modules table of an ARCHY database to determine the subordinate subroutines, provided you want them included in the FORTRAN file. Figure 7.1 shows the database tables and categories that EXTRA uses to determine the subordinate modules. The database tables are defined by a "#", table entries are defined by an "@", and the categories are distinguished by a ":".

```
#Modules<32>
@
:Called modules<32>[Modules]
```

Figure 7.1. Database Tables And Categories Used By EXTRA

b. How To Use EXTRA

1) Enter the input and output files on the ARCHY main menu screen. When EXTRA is chosen from the ARCHY Main Menu, the default extension for the input and output files is ".FOR".

2) Choose EXTRA from the main menu. Figure 7.2 shows a sample execution of EXTRA while appending the subroutines to an existing FORTRAN program and is explained below. Figure 7.6 shows the execution of EXTRA without appending the coding. The user inputs are shown within the { }.

After you choose EXTRA from the main menu, you will be asked "Do you want the code appended to an existing file [Y]?". The default to this question is "Y". If you answer this question "Yes", you will be asked "What is the name of the file to append coding to (.for will be added)==>?". Then you will be asked "Do you want the subordinate subroutines included [Y]?". The default to this question is "Y". Answer "Yes", if you would like all the subroutines called by the desired subroutine to be included in the Output file. Next you will be asked "No. of subroutines to be extracted (1-100)", here you enter the number of known subroutines you would like extracted from the Input file. You will then be asked for the names of the known subroutines you would like to have extracted. After the names of the subroutines have been entered and you are having them appended to an existing file, EXTRA will write a list of the extracted subroutines to the screen. Figure 7.3 shows the input FORTRAN file, Figure 7.4 shows the file being appended to, and Figure 7.5 shows

the FORTRAN output file after the execution of EXTRA shown in Figure 7.2.

```
------------------------- ARCHY MAIN MENU -----------------------------
INPUT FILE:  main1.RCH
OUTPUT FILE:  main3.FOR
------------------------------ EXTRA ---------------------------------
  |
  |
  |   Do you want the code appended to an existing file [Y]?
  | {Y}
  |
  |   What is the name of the file to append the coding to (.for will be added)==>
  | {main2}
  |
  |   Do you want the subordinate subroutines included [Y]?
  | {Y}
  |
  |   No. of subroutines to be extracted (1-100)
  | {1}
  |
  |   Subroutine   1 to be extracted -
  | {a}
  |
  |   The file main.for should contain the following:
  |                1  A
  |                2  B
  |                3  C
  |
  |
-----------------------------------------------------------------------
```

Figure 7.2. Sample Execution Of EXTRA While Appending

```
      program main1
      character*72 one,two,three
      one='This is'
      two='a test'
      three='of EXTRA.'
      call a(one,two,three)
      stop
      end
c*******************************************************************************
c*******************************************************************************
      subroutine a(one,two,three)
      character*72 one,two,three
      write(*,9010)one
 9010 format(1x,a72)
      call b(two,three)
      return
      end
c*******************************************************************************
c*******************************************************************************
      subroutine b(two,three)
      character*72 two,three
      write(*,9010)two
 9010 format(1x,a72)
      call c(three)
      return
      end
c*******************************************************************************
c*******************************************************************************
      subroutine c(three)
      character*72 three
      write(*,9010)three
 9010 format(1x,a72)
      return
      end
```

Figure 7.3. Input FORTRAN File

```
      program main2
      character*72 one,two,three
      one='This file has'
      two='been appended with'
      three='subroutines a,b and c'
      call a(one,two,three)
      stop
      end
```

Figure 7.4. File To Append The Coding To

```
      program main2
      character*72 one,two,three
      one='This file has'
      two='been appended with'
      three='subroutines a,b and c'
      call a(one,two,three)
      stop
      end
c*******************************************************************************
c*******************************************************************************
      subroutine a(one,two,three)
      character*72 one,two,three
      write(*,9010)one
 9010 format(1x,a72)
      call b(two,three)
      return
      end
c*******************************************************************************
c*******************************************************************************
      subroutine b(two,three)
      character*72 two,three
      write(*,9010)two
 9010 format(1x,a72)
      call c(three)
      return
      end
c*******************************************************************************
c*******************************************************************************
      subroutine c(three)
      character*72 three
      write(*,9010)three
 9010 format(1x,a72)
      return
      end
```

Figure 7.5. Output File With Coding Appended

If you answer "No" to the question "Do you want the code appended to an existing file [Y]?", you will be asked "Do you want the subordinate subroutines included [Y]?". If you answer "No", only the subroutines you input will be included in the output file. Next you will be asked for the "No. of subroutines to be extracted (1-100)" and to enter the names of the subroutines to be extracted. Figure 7.7

shows the input FORTRAN file and Figure 7.8 shows the FORTRAN output file after the execution of EXTRA shown in Figure 7.6.

```
------------------------- ARCHY MAIN MENU -----------------------------
INPUT FILE:  main1.RCH
OUTPUT FILE:  main3.FOR
---------------------------- EXTRA --------------------------------
  |
  |
  |   Do you want the code appended to an existing file [Y]?
  | {N}
  |
  |   Do you want the subordinate subroutines included [Y]
  | {N}
  |
  |   No. of subroutines to be extracted (1-100)
  | {3}
  |
  |   Subroutine   1 to be extracted -
  | {a}
  |
  |   Subroutine   2 to be extracted -
  | {b}
  |
  |   Subroutine   3 to be extracted -
  | {c}
  |
  |
```

Figure 7.6. Sample Execution Of EXTRA Without Appending

```
      program main1
      character*72 one,two,three
      one='This is'
      two='a test'
      three='of EXTRA.'
      call a(one,two,three)
      stop
      end
c*********************************************************************
c*********************************************************************
      subroutine a(one,two,three)
      character*72 one,two,three
      write(*,9010)one
```

```
 9010 format(1x,a72)
      call b(two,three)
      return
      end
c*******************************************************************************
c*******************************************************************************
      subroutine b(two,three)
      character*72 two,three
      write(*,9010)two
 9010 format(1x,a72)
      call c(three)
      return
      end
c*******************************************************************************
c*******************************************************************************
      subroutine c(three)
      character*72 three
      write(*,9010)three
 9010 format(1x,a72)
      return
      end
```

Figure 7.7. Input FORTRAN File

```
      subroutine a(one,two,three)
      character*72 one,two,three
      write(*,9010)one
 9010 format(1x,a72)
      call b(two,three)
      return
      end
c*************************************************************************
c*************************************************************************
      subroutine b(two,three)
      character*72 two,three
      write(*,9010)two
 9010 format(1x,a72)
      call c(three)
      return
      end
c*************************************************************************
c*************************************************************************
      subroutine c(three)
      character*72 three
      write(*,9010)three
 9010 format(1x,a72)
      return
      end
```

Figure 7.8. Output File Without Appending

c. Requirements For The Operation Of EXTRA

This section describes some limitations and useful information which may aid you in your operation of EXTRA.

1) Limitations

EXTRA will only allow the database to have a maximum of 50,000 lines 80 characters long. EXTRA may allocate up to 4 Megabytes of virtual memory.

2) Information

As EXTRA is executed it opens a file with the same name as the Input File with the extension ".RCH" , therefore an ARCHY database with the same name as the Input File must exist in the default directory in order to execute EXTRA. The number of subroutines you can input to be extracted can be between 1 and 100. Extra will write a note to the screen saying "Could not find a subroutine named _____ .", if it can not find a subroutine that you request. The default Output file when appending coding to an existing file is a new version of the file the coding is appended to.

8. TRAMP: Fill in Subprogram Arguments in Database

The TRAMP program is part of the ARCHY system. TRAMP fills in missing variables in the ARCHY database.

a. What TRAMP Does

TRAMP finds out which modules use a specific variable and then tramps up through the modules to determine the first place the variable could be introduced to all the modules using the variable. Then TRAMP inserts the variable in the Output ARCHY database as an internal variable in the originating subroutine and as an external variable - just passing through in the subroutines between the original and the subroutines that use the variable. TRAMP uses the Modules table of the Input ARCHY database to determine the order of the subroutines, and the Data Dictionary table to determine all the variables to check. Figure 8.1 shows the database tables and categories that TRAMP uses to determine the variables and the subroutine order. The database tables are defined by a "#", table entries are defined by an "@", and the categories are distinguished by a ":".

```
 #Modules<32>
@
:Called modules<32>[Modules]
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
#Data Dictionary<32>
@
```

Figure 8.1. Database Tables And Categories Used By TRAMP

b. How To Use TRAMP

1) Enter the input and output files on the ARCHY main menu screen. When TRAMP is chosen from the ARCHY Main Menu, the default extension for the input file is ".RCH" and the default output file is a new version of the input file.

2) Choose TRAMP from the main menu. Figure 8.2 shows a sample execution of TRAMP. The user inputs are shown within the { }.

After you choose TRAMP from the main menu, you will be asked to "Input the name of the top module ==>". The name of the top module must be entered in exactly the same way it appears in the Modules table of the Input database. The variable names are listed on the screen as TRAMP checks them. Figure 8.3 shows a sample Input ARCHY database and a sample Output ARCHY database is shown in Figure 8.4.

```
------------------------- ARCHY MAIN MENU -----------------------------
INPUT FILE:  main.RCH
OUTPUT FILE:  MAIN.RCH
------------------------------ TRAMP ---------------------------------
  |
  |    Input the name of the top module ==>
  |   {main}
  |
  |    one
  |    two
  |    three
  |
  |
  |
  |
```

Figure 8.2. Sample Execution Of TRAMP

```
#Modules<32>
@main
:Called modules<32>[Modules]
a
b
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
@a
:Called modules<32>[Modules]
:External variables - used<32>[Data Dictionary]
one
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
@b
:Called modules<32>[Modules]
c
d
:External variables - used<32>[Data Dictionary]
two
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
@c
:Called modules<32>[Modules]
:External variables - used<32>[Data Dictionary]
two
three
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
@d
:Called modules<32>[Modules]
:External variables - used<32>[Data Dictionary]
one
three
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
#Data Dictionary<32>
@one
@two
@three
```

Figure 8.3. ARCHY Database Prior To Execution Of TRAMP

```
#Modules<32>
@main
:Called modules<32>[Modules]
a
b
:External variables - used<32>[Data Dictionary]
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
one
two
@a
:Called modules<32>[Modules]
:External variables - used<32>[Data Dictionary]
one
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
@b
:Called modules<32>[Modules]
c
d
:External variables - used<32>[Data Dictionary]
two
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
one
:Internal variables<32>[Data Dictionary]
three
@c
:Called modules<32>[Modules]
:External variables - used<32>[Data Dictionary]
two
three
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
@d
:Called modules<32>[Modules]
:External variables - used<32>[Data Dictionary]
one
three
:External variables - set<32>[Data Dictionary]
:External variables - just passing through<32>[Data Dictionary]
:Internal variables<32>[Data Dictionary]
```

```
#Data Dictionary<32>
@one
@two
@three
```

Figure 8.4. ARCHY Database After Execution Of TRAMP

Figure 8.5 shows a structure chart diagram of the Modules table from the database shown in Figure 8.3 . The diagram also lists the variables for each module prior to the execution of TRAMP. None of the variables are listed as internal variables or as external variables passing through. Variables ***one***, ***two***, and ***three*** are listed as used external variables in the modules.

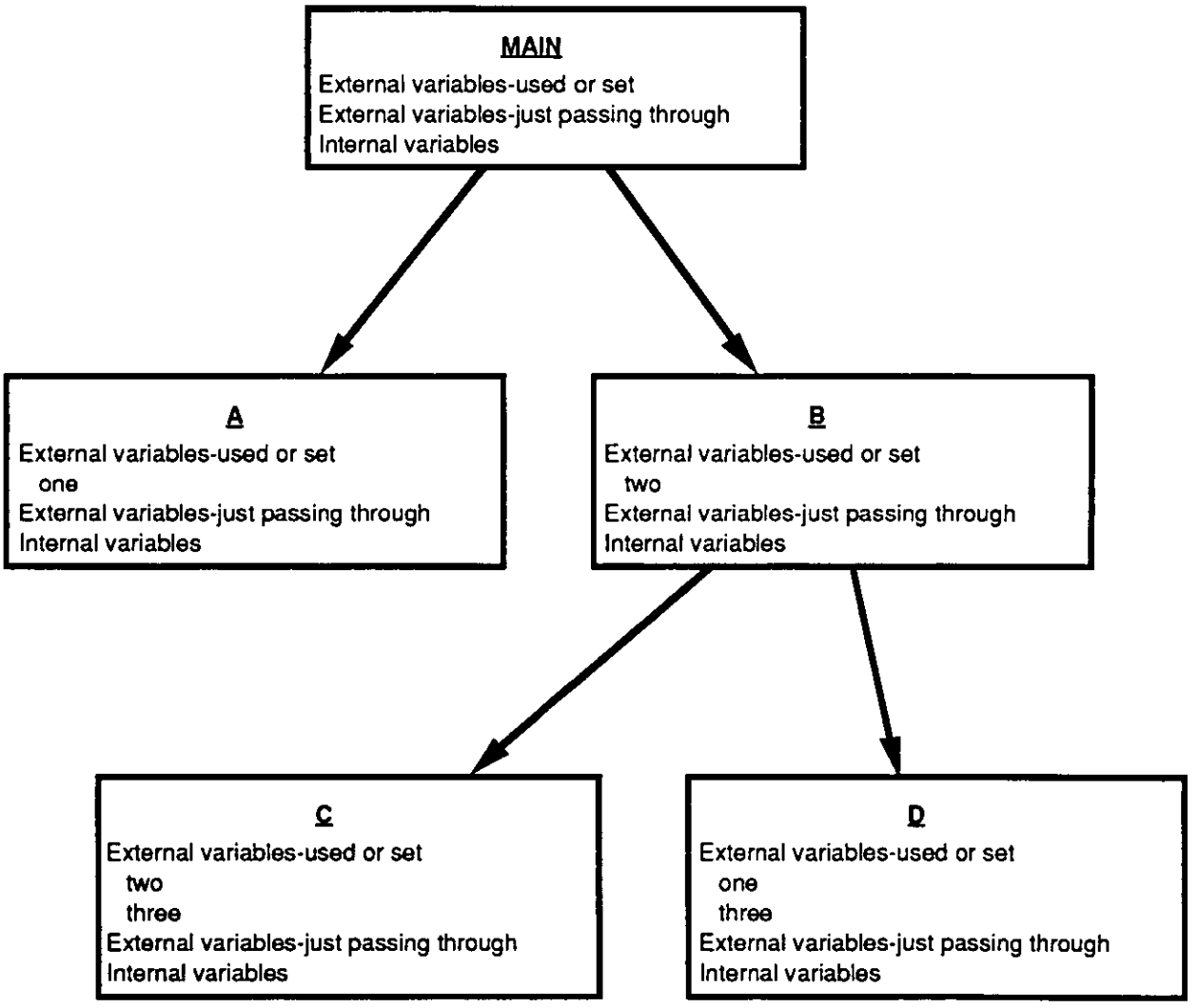


Figure 8.5. Structure Diagram Of The Input ARCHY Database

Figure 8.6 shows a structure chart diagram of the Modules table from the database shown in Figure 8.4 . The diagram also lists the

variables for each module after the execution of TRAMP. As TRAMP executes it works down the Data Dictionary table and checks on the variables. As TRAMP works with the variable ***one*** it finds the variable in modules A and D and that module MAIN is the place where the two flow paths meet. The variable ***one*** is then recorded as an internal variable in MAIN and as a external variable - just passing through in B. For the variable ***two***, TRAMP finds it in modules B and C and traces their path back to Module MAIN. The variable ***two*** is then listed in the database as an internal variable in the module MAIN, but is not listed as an external variable - just passing through in B because it is already listed as an external variable in the Input ARCHY database. TRAMP finds the variable ***three*** in modules C and D and their paths meet at module C. TRAMP places the variable ***three*** in module B as an internal variable.

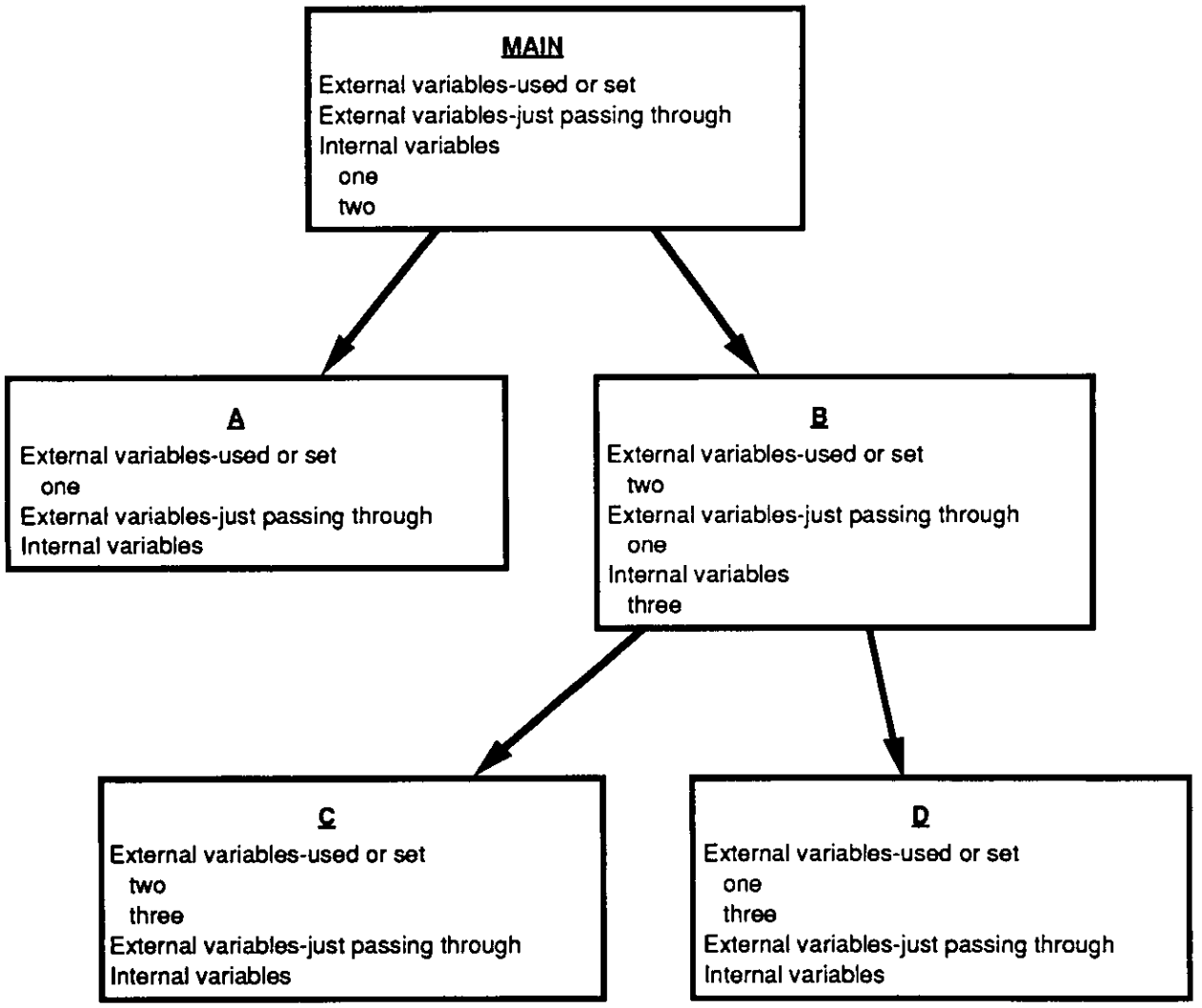


Figure 8.6. Structure Diagram Of The Output ARCHY Database

c. Requirements For The Operation Of TRAMP

This section describes some limitations and useful information which may aid you in your operation of TRAMP.

1) Limitations

TRAMP will only allow the database to have a maximum of 50,000 lines 80 characters long. The maximum number of modules allowed in the database is 100, and each module may contain 500 internal variables, 500 external variables, and 500 passed variables. The name of the top module may have up to 20 characters , otherwise TRAMP will not execute correctly. TRAMP may allocate up to 20 Megabytes of virtual memory.

2) Information

TRAMP will not execute correctly if the name of the top module is entered differently than it appears in the Modules table of the Input ARCHY database. Also, the variables can not be listed as external variables in the Input database listing for the top module or TRAMP will not operate correctly. The Data Dictionary table of the Input database must contain the variables in order for TRAMP to check on the variables. TRAMP only lists the variable as an external variable - just passing through if it is not already listed as an external variable - used or set.

B. Beautify Menu

1. CLEAN: Indent and Resequence Statement Numbers

This chapter tells you how to use CLEAN which is one of the programs used to blow the dust off "ancient" FORTRAN programs so that they can be more readable and maintainable.

a. What CLEAN Does

CLEAN reads a FORTRAN program and outputs the same program with the following features. CLEAN automatically indents the body of DO loops and IF-THEN-ELSE statements. It resequences statement labels within subprograms. It converts Hollerith constants within FORMAT statements to literals enclosed by single quotes. It converts executable statements to lower-case except for portions enclosed in quotes. It inserts two lines of stars in comment statements between subprograms. It changes TABs in columns 1-7 to spaces. If you want it to, it will place FORMAT statements at the end of each subprogram in the order in which they are encountered and it will put a frame of stars around all comment statements. CLEAN is guaranteed to produce a program that executes exactly like the program that was fed to it provided that the source program does not contain any lines that have trailing blanks within literals which are continued on the next line.

b. How To Use CLEAN

1) Enter the input file (name of program being cleaned) on the ARCHY main menu screen as shown in Figure B.1.1.

2) Choose BEAUTIFY from the main menu. ".FOR" is the default extension when BEAUTIFY is chosen from the main menu. The default output file is a new version of the program with the same name and extension.

3) Choose CLEAN from the BEAUTIFY menu. As CLEAN processes your program you see a list that tells the number of lines in each module. Then you see each module name on the screen. When the CLEAN program is finished you see the BEAUTIFY menu once more.

Note: The default preferences for CLEAN are for it to put a frame of stars around comments and to place FORMAT statements at the end

of subroutines. Refer to page 8 to find out how to change these default settings.

```
------------------------- ARCHY MAIN MENU -----------------------------
INPUT FILE:  main.FOR
OUTPUT FILE: MAIN.FOR
  ------------------------ BEAUTIFY -----------------------------
  |  ---------------------- CLEAN --------------------------
  | |
  | |
  | |      Line no. =        10
  | |      Line no. =        20
  | |      Line no. =        41
  | |
  | | END OF ORIGINAL CODING
  | |    2subroutine a
  | |     3subroutine b
  | |
  | |
  | | END OF ORIGINAL CODING
  | |
```

Figure B.1.1. Sample CLEAN Execution

Figures B.1.2 and B.1.3 show what a program looks like before and after cleaning.

```
       PROGRAM MAIN
C      EXAMPLE PROGRAM FOR CLEAN
50     GOTO 40
230    FORMAT(1X,'THE END')
40     DO 20 I = 1,3
20     IF (I.EQ.2)GOTO 7893
7893   WRITE(8,230)
       CALL B
       STOP
       END
       SUBROUTINE B
40     IF (.TRUE.) THEN
35     IF (1=1) THEN
       A=2
       ELSE
       A=3
       END IF
       ELSE
       A=4
       ENDIF
       IF (.TRUE.) THEN
       A=5
       ELSE
       IF (.TRUE) THEN
       B=3
       ELSE
       D=4
       ENDIF
       ENDIF
       RETURN
       END
```

Figure B.1.2. TEST1.FOR Before Cleaning

```
      program main
c*************************************************************************
c
c     EXAMPLE PROGRAM FOR CLEAN
c
c*************************************************************************
10    goto 20
20    do 30 i = 1,3
        if (i.eq.2)goto 40
30    CONTINUE
40    write(8,9010)
      call b
      stop
9010  format(1x,'THE END')
      end
c*************************************************************************
c*************************************************************************
      subroutine b
  10  if (.true.) then
  20    if (1=1) then
          a=2
        else
          a=3
        end if
      else
        a=4
      endif
      if (.true.) then
        a=5
      else
        if (.true) then
          b=3
        else
          d=4
        endif
      endif
      return
      end
```

Figure B.1.3. TEST1.FOR After Cleaning

c. How CLEAN Handles Special Cases

This section describes some of the special features of CLEAN in the detail that is appreciated by programmers but may be tedious to the casual reader. If you trudge through it you may become an avid ARCHYologist.

1) Limitations of Space

You can only have 500 subprograms in your entire program. Each of those subprograms can have at most 499 statement labels of which no more than 99 may be statement labels for FORMAT statements. CLEAN stores all the statement labels for an entire program in an array so it could conceivably need to reference an array of 249,500 5-byte character strings. Lines must be no more than 80 characters and end-of-line labels in columns 73-80 are deleted by CLEAN but another program on the BEAUTIFY menu, LABEL, will generate line labels using the first 4 characters of the subprogram name.

2) Comment Lines

i. To Frame or Not To Frame?

Nothing is as distinctive about your programming style as your comments so if you prefer to not have stars around your comments then the only stars CLEAN will insert in your program are the two lines of stars between subprograms. See page 8 for an explanation of how to set preferences. The program from Figure B.1.1 turns out like Figure B.1.4 if you choose not to frame. Please note that once comment lines are added to a program by CLEAN, they are never taken away by CLEAN with one exception. CLEAN deletes duplicate comment lines that consist of a c followed by 72 blanks. This is to keep frames from being bigger than they need to be. When you choose not to frame comments they are padded with blanks so that all comment lines become 80 characters long.

```
      program main
c     EXAMPLE PROGRAM FOR CLEAN
   10 goto 20
   20 do 30 i = 1,3
        if (i.eq.2)goto 40
   30 CONTINUE
   40 write(8,9010)
      call b
      stop
9010 format(1x,'THE END')
      end
c*******************************************************************************
c*******************************************************************************
      subroutine b
   10 if (.true.) then
   20   if (1=1) then
          a=2
        else
          a=3
        end if
      else
        a=4
      endif
      if (.true.) then
        a=5
      else
        if (.true) then
          b=3
        else
          d=4
        endif
      endif
      return
      end
```

Figure B.1.4. Comments Are Not Framed

3) End-Of-Line Comments

VAX FORTRAN permits an extension to the FORTRAN-77 standard called end-of-line comments. The VAX FORTRAN compiler ignores characters that occur to the right of an exclamation point. CLEAN

converts end-of-line comments to a C-type comment with the comment occurring one line previous to where the end-of-line comment had been. This is illustrated by Figures 5 and 6.

```
      PROGRAM TESTCOM
      CHARACTER*10 ACHAR
      ACHAR='SURPRISE' !END OF LINE COMMENT
      ACHAR='THE!END'
      STOP
      END
```

Figure B.1.5. Program TESTCOM Before Cleaning

```
      program testcom
       character*10 achar
c                          END OF LINE COMMENT
       achar='SURPRISE'
       achar='THE!END'
       stop
       end
```

Figure B.1.6. Program TESTCOM After Cleaning

4) Indenting and Continuation Lines

The bodies of nested loops and conditionals are indented two spaces. When the level of nesting exceeds 30, CLEAN indents no more as shown in Figure B.1.7. Continuation lines will normally be indented two spaces to the right of the first line of a continued statement. The location of line breaks within a continued statement are normally preserved by CLEAN. The exceptions to these rules occur when indention would cause a line to be longer than 72 characters in which case the overflow is continued on the next line. CLEAN never inserts line breaks in the middle of keywords or variables. If a line break falls in the middle of a literal then the continuation line is not indented. Literals that have to be continued on the next line are never written with trailing blanks at the end of the line that is being

continued. When CLEAN reads a line it starts at column 72 and looks at each character from right to left. Any blank characters are trimmed. If a program is fed to CLEAN with a literal containing trailing blanks which is split across two lines, then CLEAN's output when compiled and executed could result in blanks being omitted from the program's output which could lead to confusing output as shown in Figures 8 and 9. In Figure B.1.8 the literal in line 10 is continued across continuation lines with trailing blanks but after it is put through CLEAN it looks like Figure B.1.9.

```
      program test7
        do i=1,10
          do i1=1,10
            do i2=1,10
              do i3=1,10
                do i4=1,10
                  do i5=1,10
                    do i6=1,10
                      do i7=1,10
                        do i8=1,10
                          do i9=1,10
                            do i10=1,10
                              do i11=1,10
                                do i12=1,10
                                  do i13=1,10
                                    do i14=1,10
                                      do i15=1,10
                                        do i16=1,10
                                          do i17=1,10
                                            do i18=1,10
                                              do i19=1,10
                                                do i20=1,10
                                                  do i21=1,10
                                                    do i22=1,10
                                                      do i23=1,10
                                                        do i24=1,10
                                                         do i25=1,10
                                                         do i26=1,10
                                                         do i27=1,10
                                                         do i28=1,10
                                                         do i29=1,10
                                                         do i30=1,10
                                                         do i31=1,10
                                                         do i32=1,10
                                                         print *,
      *                                                   'LOOP DE LOOP'
                                                         enddo
                                                         enddo
                                                         enddo
                                                         enddo
                                                         enddo
                                                         enddo
                                                         enddo
                                                         enddo
                                                        enddo
                                                      enddo
                                                    enddo
```

```
                                                    enddo
                                                  enddo
                                                enddo
                                              enddo
                                            enddo
                                          enddo
                                        enddo
                                      enddo
                                    enddo
                                  enddo
                                enddo
                              enddo
                            enddo
                          enddo
                        enddo
                      enddo
                    enddo
                  enddo
                enddo
              enddo
            enddo
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
stop
end
```

Figure B.1.7. Nesting Level Exceeded

```
      PROGRAM TRAIL
C      Y=TEMPERATURE, OUJ=MEASUREMENT, ER=ENERGY, K=CONSTANT
       WRITE(6,10)
10     FORMAT(1X,'Y
      *OUJ
      *ER
      *K')
       WRITE(6,20)
20     FORMAT(1X,'-     ---     --     -')
       STOP
       END


Y     OUJ     ER     K
-     ---     --     -
```

Figure B.1.8. Program and Output Before Cleaning

```
      program trail
      Y=TEMPERATURE, OUJ=MEASUREMENT, ER=ENERGY, K=CONSTANT
      write(6,9010)
      write(6,9020)
      stop
9010  format(1x,'YOUJERK')
9020  format(1x,'-     ---     --     -')
      end


YOUJERK
-     ---     --     -
```

Figure B.1.9. Program and Output After Cleaning

5) Hollerith Conversion

Hollerith constants within FORMAT statements are converted to literals whereas Hollerith constants elsewhere are left unperturbed. The one exception to this rule is a Hollerith constant within a FORMAT statement that contains a literal. These statements are flagged with an H in the first column and CLEAN beeps and prints an error message. If the offending FORMAT statement happens to be continued as well then the continuation of this line will not be marked with an H which could lead to unpredictable results if the program even compiles. If you get this message then the original program must be edited so that Hollerith constants within FORMAT statements which contain literals are removed. Examples of Hollerith constant conversion to literals is shown in Figures 10 and 11.

```
      PROGRAM TEST12
200   FORMAT(1X,1HE,5HTWICE,6HTHRICE)
2100  FORMAT(1X,12HERROR!AGAINS)
      STOP
      END
```

Figure B.1.10. Program TEST12 Before Cleaning

```
      program test12
      stop
9010  format(1x,'E','TWICE','THRICE')
9020  format(1x,'ERROR!AGAINS')
      end
```

Figure B.1.11. Program TEST12 After Cleaning

2. ALPHA: Alphabetize Modules

This program puts the modules within a program file into alphabetical order. The preference file determines whether the main program appears at the top of the file or in alphabetic order with the other modules. How to set preferences is explained on page 8 above.

3. LABEL: Add End-of-Line Labels

This program adds line labels to column 72-80 of each executable line. The labels consist of the first 4 characters of the module line followed by a 4 digit number, The 4 digit numbers are ordered sequentially with an increment of 10. If you have more than 999 executable lines in a module then the numbering begins again at 0. Figure B.3.1 shows a program before LABEL is run and Figure B.3.2 shows the same program after LABEL is run. It is poor programming practice to continue a literal on another line. Notice that if you continue a literal on another line then the line label will be inserted in the middle of the literal.

```
      program aaaaaa
c      comments are not labeled
       character*80 a
       a='label test'
       a=
      1'Continued lines get labeled too'
       a='Watch out for
      1quotes.'
       stop
       end
       subroutine bbbbbb
c      comments are not labeled
       character*80 a
       a='label test'
       return
       end
       subroutine ccc
c      comments are not labeled
       character*80 a
       a='label test'
       return
       end
```

Fig. B.3.1 Program Before LABEL is Run

```
      program aaaaaa                                              aaaa  10
      comments are not labeled
      character*80 a                                              aaaa  20
      a='label test'                                              aaaa  30
      a=                                                          aaaa  40
     1'Continued lines get labeled too'                           aaaa  50
      a='Watch out for                                            aaaa  60
     1quotes.'                                                    aaaa  70
      stop                                                        aaaa  80
      end                                                         aaaa  90
      subroutine bbbbbb                                           bbbb  10
      comments are not labeled
      character*80 a                                              bbbb  20
      a='label test'                                              bbbb  30
      return                                                      bbbb  40
      end                                                         bbbb  50
      subroutine ccc                                              ccc   10
      comments are not labeled
      character*80 a                                              ccc   20
      a='label test'                                              ccc   30
      return                                                      ccc   40
      end                                                         ccc   50
```

Fig. B.3.2 Program After LABEL is Run

4. PORT: Convert File for Transmission to IBM

This program reads a file using and then writes it back out using the parameter, CARRIAGECONTROL=NONE. At the same time a blank is added at the beginning of each line. This is the preferred format for a program prior to uploading of that file to the IBM mainframe.

5. PRETRN: Insert Blank Before Each Line

This program inserts a blank at the beginning of each line. This is done to facilitate printing of the file because many printers drop the first character on each line.

C. Output Menu

1. STRUCT: Structure Chart

When option 1 on the output menu is chosen, a structure chart is printed for the ARCHY database that was entered on the main menu as the input file. The PostScript commands that draw the structure chart are output to a file with the default extension, SPS, and then that file is submitted to the print queue shown on the output menu screen. An example of a structure chart is shown in Figure C.1.1.

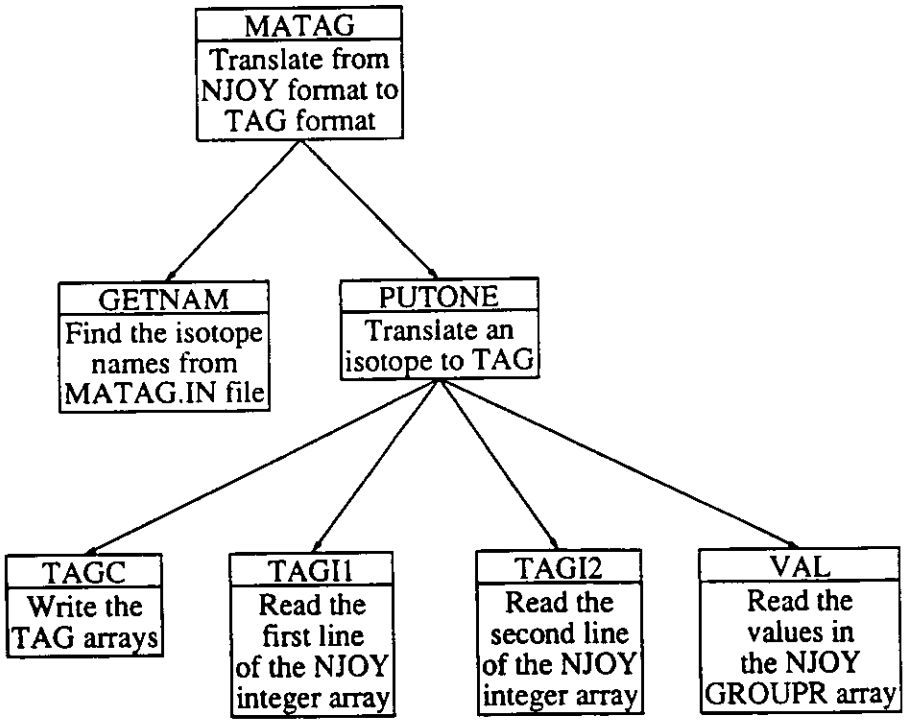


Fig. C.1.1 Sample Structure Chart

2. HEADER: Module Design Sheets

When option 2 on the output menu is chosen, module design sheets are printed for the ARCHY database that was entered on the main menu as the input file. Module design sheets contain the same descriptive information found in a module header. An example of a module design sheet is shown in Figure C.2.1. The module design sheets are output to a file with the default extension, MDS, and then that file is submitted to the print queue shown on the output menu screen.

MODULES Descriptions

* * * * *

Module name:
MATAG
Structure chart description:
Translate from
NJOY format to
TAG format
Purpose:
Translate from NJOY format to TAG format
History:
Designed: R. E. Pevey and K. R. Okafor 8/17/90
Coded: R. E. Pevey 8/18/90
Called modules:
getnam
putone
Internal variables:

ACHOICE Flag for real/binary (=r/b) output for TAG file	C*1
IEND Flag that last entry has been read from NAMES file	i*4
IMAT Material number	i*4
JOSDES (999) JOSHUA isotope/temperature name	C*13
MATDES (999) NJOY material name (Must be a .GROUPR file of this name)	C*13
NMAT Number of material pairs read	i*4

Variables written:

JOSDES (999) JOSHUA isotope/temperature name	C*13
MATDES (999) NJOY material name (Must be a	C*13

```
        .GROUPR file of this name)

Algorithm:
  1. Get the NJOY and JOSHUA names of isotopes desired [GETNAM]
  2. Loop over isotopes desired:
     A. Translate the isotope [PUTONE]
```

Fig. C.2.1 Sample Module Design Sheet

3. DATADICT: Design Sheet Data Dictionary

When option 3 on the output menu is chosen, a design sheet data dictionary is printed for the ARCHY database that was entered on the main menu as the input file. The design sheet data dictionary is output to a file with the default extension, DDD, and then that file is submitted to the print queue shown on the output menu screen. An example of a design sheet data dictionary is shown in Figure C.3.1.

DATA DICTIONARY Descriptions

* * * * *

Variable name:
ACHOICE
Type:
C*1
Definition:
Flag for real/binary (=r/b)
output for TAG file
Used by:
matag
putone
tagc

* * * * *

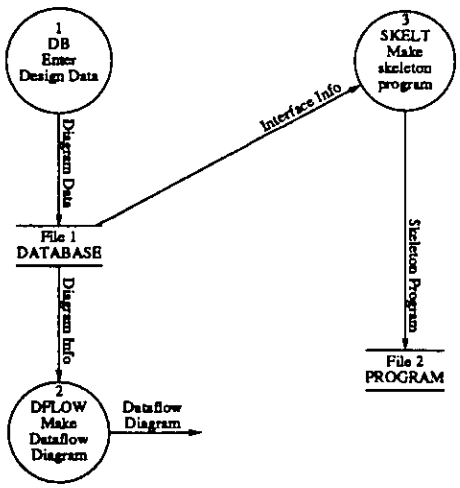
Variable name:
ALINE
Type:
C*80
Definition:
Variable to hold latest line
read from a file
Used by:
getnam
putone
tagi1
tagi2
val

Fig. C.3.1 Sample Design Sheet Data Dictionary

4. DATAFLOW: Dataflow Diagram

When option 4 on the output menu is chosen, a dataflow diagram is printed for the ARCHY database that was entered on the main menu as the input file. The PostScript commands that draw the dataflow diagram output to a file with the default extension, FPS, and then that file is submitted to the print queue shown on the output menu screen. An example of a dataflow diagram is shown in Figure C.4.1.

Fig. C.4.1 Sample Dataflow Diagram



5. FLOWDICT: Dataflow Diagram Data Dictionary

When option 5 on the output menu is chosen, dataflow diagram data dictionaries are printed for the ARCHY database that was entered on the main menu as the input file. The output goes to a file with the default extension, FDD, and then that file is submitted to the print queue shown on the output menu screen. An example of a dataflow diagram data dictionary is shown in Figure C.5.1.

DFD DATA DICTIONARY

MATERIAL NUMBER
  Variables
    IMAT I*4
      Material number

ISOTOPE NAMES
  Variables
    NMAT I*4
      Number of material pairs read
    JOSDES (999) C*13
      JOSHUA isotope/temperature name
    MATDES (999) C*13
      NJOY material name (Must be a .GROUPR file of this name)

FIRST LINE OF TAG
  Variables
    MAT I*4
      MAT number (isotope designation)
    MF I*4
      MF file number
    MT I*4
      NJOY MT (reaction type) value
    NL I*4
      Legendre order

Fig. C.5.1 Sample Dataflow Diagram Data Dictionary

6. TRANFORM: Transform Descriptions

When option 6 on the output menu is chosen, transform descriptions are printed for the ARCHY database that was entered on the main menu as the input file. When this program runs, you are asked whether you want to print the transform descriptions for all the dataflow diagrams within the database or whether you only want one of them. You indicate your choice by typing "a" or "o" respectively. The transform descriptions are output to a file with the default extension, TDD, and then that file is submitted to the print queue shown on the output menu screen. An example of a transform description is shown in Figure C.6.1.

```
                    TRANSFORM DESCRIPTIONS

Transform  - MATAG
  Title
      Translate from
      NJOY format to
        TAG format
  Description
      Translate from NJOY format to TAG format
  Procedure
      1. Get the NJOY and JOSHUA names of isotopes desired [GETNAM]
      2. Loop over isotopes desired:
         A. Translate the isotope [PUTONE]
  Incoming data flows
      NJOY MULTIGROUP RECORD
      LIST OF DESIRED ISOTOPES
  Outgoing data flows
      TAG MULTIGROUP RECORD
  Subordinate transforms
      GETNAM
      PUTONE
```

Fig. C.6.1 Sample Transform Description

APPENDIX A

I.Structure of the ARCHY Database

Here is the latest version of the ARCHY database structure; the first version appeared in the ARCHY PSSP, but has undergone some variation since then. The bracketed information designates which Tables may be linked to the current category. This remains so that it is possible to eliminate an entry in a Table and at the same time eliminate all references to it (i.e., all possible module names would contain a [Modules] designation).

Here is a definition of each of the tables and a listing of its fields:

(1) Module table - Individual modules are described. <32>

Fields:

Structure chart description <16> - The title on the Structure Chart
Purpose <64> - Description of the purpose of the module
History <64> - Description of history of module
Called Modules <32> [Modules] - Listing of other modules called by this one
Parameters {value} <64> - Module parameters and their values
External variables - used <32> [Data Dictionary] - List of variables which are used in the module
External variables - set <32> [Data Dictionary] - List of variables which are set within the module
External variables - just passing through <32> [Data Dictionary] - List of variables which are neither set nor used, but simply pass through on the way to a subordinate
Internal variables <32> [Data Dictionary] - List of internal variables (i.e., not passed from above, but may be passed below)
Theory <64> - Theoretical basis for the algorithm followed with references to technical literature.

Algorithm <64> - Outline of the algorithm followed in the module

(2) Data Dictionary table <32>

Fields:
Type - <16> Type information
Array limits <32> - Array information
Description <62>
Modules <32> [Modules] - Only needed in the case of duplicate variable names

(3) COMMON block table <32>

Fields:
Description <64>
Modules <32> [Modules]
Variable names <32> [Data Dictionary] - A list of variables in the COMMON Block

(4) Transform table - This table contains the information for each transform of the Data Flow Diagram<32>

Fields:
Transform number <16> - A numerical designation to appear on the DFD
Transform title <32> - Transform title on the DFD
Description <64>
Procedure <64>
Incoming data flows <64> - Lists the flows containing the input for the transform
Outgoing data flows <64> - Lists the flows containing the output from the transform
Subordinate transforms <32> [Transforms]- Lists the transforms (if any) into which this one can be broken down

Corresponding Modules <32> [Module] - This lists the module(s) that implement the transformation

(5) Files table - Here the Data Flow Diagram files (input and output data repositories) are described. <32>

Fields:
File number <16> - A DFD numerical designation such as "File 7"
File title <32> - This will be the title of the file on the DFD
Description <64>
Incoming data flows <64> [Data flows] - Lists the flows containing the input for the data repository
Outgoing data flows <64> [Data flows] - Lists the flows containing the output from the data repository

(6) Data Flow table - Data flows are described<64>

Fields:
Description <64>
Variables <32> [Data dictionary] - A list of variables in the data flow

(7) DFD table - The individual DFD drawings are organized. <64>

Fields:
DFD Title <64> - This will show up as the title of the DFD
Description <64>
Displayed transforms{x,y,angin,angout} <32> - [Transforms] The transforms listed will be printed at the locations given (x and y should range from 0 to 1). All data flows associated with them will be included also. Any input and output dataflows which have no displayed "other end" will be attached at the angles indicated.

Displayed files {x,y} <32> [Files]
The files listed will be printed at the locations given (x and y should range from 0 to 1).

Extra incoming dataflows <64> - [Data flows]
Extra incoming data flows desired on the drawing.

Extra outgoing dataflows <64> - [Data flows]
Extra outgoing data flows desired on the drawing.

Legend position <32> The x and y locations of the Legend using the format "(x,y)". (No entries results in no Legend.)