FLOPPY and FLOW User Guide
VERSION 6

```
 FFFFFFF  LL         OOOOOO      PPPPPP     PPPPPP     YY     YY
FFFFFFFF  LL        OOOOOOOO    PPPPPPPP   PPPPPPPP   YY     YY
FF        LL        OO    OO    PP    PP   PP    PP   YY     YY
FFFFFF    LL        OO    OO    PP    PP   PP    PP    YY   YY
FFFFFF    LL        OO    OO    PPPPPPPP   PPPPPPPP     YYYY
FF        LL        OO    OO    PPPPPPP    PPPPPPP       YY
FF        LL        OO    OO    PP         PP            YY
FF        LL        OO    OO    PP         PP            YY
FF        LLLLLL    OOOOOOOO    PP         PP            YY
FF         LLLLLL    OOOOOO     PP         PP            YY
```

F L O P P Y – CODING CONVENTION CHECKER

F L O W – STRUCTURE ANALYSERS

!!! WARNING ... CODING CONVENTIONS MAY IMPROVE YOUR FORTRAN

VERSION 6.00 (May 1990) (Unix support added)

July 28, 1992

******** Sixth Edition ********

******** J.J. Bunn ********

CERN/DD/US/112

Contents

Chapter 1

Introduction

This document describes the use of FLOPPY and FLOW on the VAX/VMS, IBM/CMS and MIPS/Ultrix machines in the CERN Computer Centre. FLOPPY is a software tool that takes as input a file of FORTRAN code and then checks it according to various "coding conventions" that have been defined. It can also "tidy" the source FORTRAN, producing a new file with indented DO–loops, block IF–s, and so on. In addition, FLOPPY can be asked to write a binary summary file (which describes the structure of the source FORTRAN) that may then be used as input to the FLOW program. FLOW is now also available for Unix.

The FLOW program takes the binary summary file produced by FLOPPY, and, according to the wishes of the user, produces various reports on the structure of the original FORTRAN program.

In summary, FLOPPY

♣ checks FORTRAN coding conventions,
♣ "tidies" FORTRAN source,
♣ produces a summary file for FLOW.

And the FLOW program

♣ produces various reports on the structure of FORTRAN code,
♣ allows an interactive exploration of the structure,
♣ shows COMMON block variable usage.

Please note that FLOPPY is based on FLOP[1] (FORTRAN Language Orientated Parser).

Chapter 2

FLOPPY

Have you ever wondered whether the FORTRAN77 code you are writing conforms to accepted coding conventions ? Even if not, you may still be interested in using FLOPPY to see if it does. Before using FLOPPY to check your FORTRAN, you should compile the source form (preferably with the ANSI switch enabled), and verify that there are no compilation errors. If there are, then the results from using FLOPPY will be unreliable. FLOPPY treats illegal FORTRAN statements (such as 'INCLUDE' and HISTORIAN directives) as comments, and will ignore them completely. FLOPPY processes the FORTRAN routine by routine and statement by statement. When all routines have been processed, various global checks are made (e.g. for EXTERNAL functions that are not declared). One useful by–product of this treatment is the identification of COMMON blocks declared in subprograms where they are not used. FLOPPY is quite quick, particularly for short programs where the number of checks is disproportionately smaller than for larger ones. It is recommended to use FLOPPY for short programs, early on in the development cycle, rather than submit very large programs later. In this way the number of corrections that will need to be made at any time will be smaller, and thus easier to come to terms with!

---

1 H.Grote "FLOP User's Guide and Reference Manual", DD/US/13 (1985)

2.1 Warnings from FLOPPY

When a FORTRAN statement which does not conform to the conventions is identified by FLOPPY, a warning message is printed describing the error, together with the offending statement. Figure 1 shows some warnings generated by FLOPPY.

```
!!! WARNING ... NOT ENOUGH (<3) COMMENT LINES AT START OF MODULE

     8.        IF(IF(L).EQ.1) GOTO66
!!! STATEMENT HAS NO BLANK AFTER "GOTO"

    16.        WRITE(*,21) J
!!! STATEMENT SHOULD NOT HAVE LUN=*

    21.        STOP
!!! STATEMENT SHOULD BE PRECEDED BY A "WRITE"

    22.  999  END
!!! STATEMENT SHOULD NOT HAVE LABEL

!!!   5 STATEMENT WARNING(S) IN THIS MODULE

BEGIN GLOBAL CHECKS WITHIN THIS MODULE

!!! MIXED MODE EXPRESSION (BAD OPERATOR IS MARKED)
IF(I*2.0) = 3
    ×
!!! WARNING ... VARIABLE IF       CLASHES WITH FORTRAN KEY-WORD IF

!!!   2 GLOBAL WARNING(S) IN THIS MODULE
```

Figure 1: Warnings produced by FLOPPY

By default, only the statements at fault are printed, but optionally you may specify that all FORTRAN statements are printed out (this is useful for programs where the context of the statement may clarify the error). FLOPPY will only print subprogram headers for those subprograms in which errors have occured. All output from FLOPPY may either be viewed directly at the terminal or, optionally, sent to a disk file for inspection at a later time.

2.2 Conventions that may be checked by FLOPPY

Here follows the list of coding conventions which may at present be checked by FLOPPY. The conventions in the STANDARD set are marked by an asterisk (*).

```
*  1   Avoid comment lines after end of module
*  2   End all program modules with the END statement
*  3   Declared COMMON blocks must be used in the module
*  4   COMPLEX and DOUBLEPRECISION vars at end of COMMON
*  5   COMMON block definitions should not change
*  6   Variable names should be 6 or fewer characters long
```

```
   7   Variables in COMMON should be 6 characters long
   8   Variables not in COMMON should be <6 characters
*  9   Integer variables should begin with I to N
*  10  Variable names should not equal FORTRAN keywords
*  11  Avoid comment lines before module declaration
*  12  Module names should not equal intrinsic functions
*  13  First statement in a module should be declaration
*  14  Module should begin with at least 3 comment lines
   15  Comment lines should begin with a C
*  16  No comment lines between continuations
*  17  Avoid non-standard variable types eg INTEGER*2
*  18  Avoid multiple COMMON definitions per line
*  19  Do not dimension COMMON variables outside COMMON
*  20  Avoid embedded blanks in variable names
*  21  Avoid embedded blanks in syntactic entities
*  22  Avoid the use of PRINT statements (use WRITE)
   23  Do not give the END statement a label
*  24  Avoid WRITE(* construction
   25  Avoid WRITE statement in a FUNCTION
*  26  Avoid the use of PAUSE statements
*  27  Statement labels should not begin in column 1
*  28  Always preceede STOP by a descriptive WRITE
*  29  Avoid the use of ENTRY in FUNCTIONS
*  30  Avoid using I/O in FUNCTIONs
   31  Avoid the use of the alternate RETURN statement
*  32  COMMON block names should not equal variable names
*  33  Avoid use of obsolete CERN library routines
   34  Avoid FUNCTION names the same as intrinsics
*  35  Local functions should be declared EXTERNAL
*  36  Module names should all be different
*  37  Avoid expressions of mixed mode eg A=B/I
*  38  Length of passed CHARACTER variables should be *
*  39  Order of statements should conform !
*  40  Separate Statement Functions by comment lines
*  41  No names in Statement Function definitions elsewhere
   42  Use LLT,LGT etc to compare CHARACTER vars. in IFs
   43  Variables (not COMMON, not PARAMs) <6 characters
*  44  Passed arguments should be dimensioned * in module
```

2.3 Names to Ignore

Occasionally your FORTRAN may contain references to subroutines or variables over which you have no control (e.g. GEANT variables). The warnings associated with these variables or subroutines are therefore rather academic. FLOPPY thus allows you to suppress the warnings associated with either particular variables, or whole subroutines. This is done by giving FLOPPY a list of 'names to ignore' (you may specify up to fifty variable names and fifty subroutine names). The file containing the list of names to ignore is saved on disk, and may be used again in a future FLOPPY run. As an example, you may have a VMS file called TEST.FOR which contains various subroutines, amongst them SUBROUTINE OPAL, and references to the 6–character variables RINPAN and DRIDER, which do not appear in a COMMON block and therefore should be less than 6 characters long (if you follow this particular convention). You are aware of this error and wish to suppress those warnings from FLOPPY. You also want to suppress all warnings from SUBROUTINE OPAL for other reasons. You type the following FLOPPY command:

```
FLOPPY TEST.FOR /IGNORE=(#OPAL,RINPAN,DRIDER)
```

The process on CERN VM/CMS is similar; the names may be specified in the panel or on the command line.

2.4 Special Processes

At present there are two 'special processes' available in FLOPPY.

- ♣ The process ALEPH will cause the set of coding conventions to change from the STANDARD set to the ALEPH set.
- ♣ The process GALEPH (the ALEPH Monte Carlo program) is intended for people using FLOPPY to check programs that reference GEANT3 variables; it essentially causes FLOPPY to ignore variable names beginning 'G.....' or 'IG....'.

2.5 Tidying your FORTRAN

FLOPPY allows you to 'tidy' your FORTRAN in the following ways:

1. *GROUPF (IBM/CMS,VAX/VMS) –F (UNIX)*
   Group all FORMAT statements at the end of each subprogram.
2. *INDENT (IBM/CMS,VAX/VMS) –J (UNIX)*
   Indent DO–loops and IF–clauses.
3. *GOTOS (IBM/CMS,VAX/VMS) –G (UNIX)*
   Tabulate GOTOs to the right hand side of the source form.
4. *STMNTS (IBM/CMS) RENUMS (VAX/VMS) –S (UNIX)*
   Renumber all statement labels.
5. *RENUMF (IBM/CMS) FORMAT VAX/VMS) –R (UNIX)*
   Renumber all FORMAT labels.

In the case where you choose to use any of these options, a new FORTRAN file will be written containing the changes. Figure 2 shows the result of tidying a small subroutine using FLOP.

```
      Before tidying                       After tidying
      --------------                       -------------

      PROGRAM BAD                          PROGRAM BAD
      DIMENSION IF(10)                     DIMENSION IF(10)
      I=5                                  I=5
 66   J=7                               10 J=7
      DO 43 K=1,I                          DO 30 K=1,I
      IF(K-J.EQ.2) THEN                       IF(K-J.EQ.2) THEN
      DO 11 L=1,2                                DO 20 L=1,2
      IF(IF(L).EQ.1) GOTO66                         IF(IF(L).EQ.1)            GOTO 10
  21  FORMAT(1X,I2)                     20       CONTINUE
  11  CONTINUE                                                                GOTO 10
      GOTO 66                                 ELSE
      ELSE                                       IF(IF(K).EQ.IF(J)) THEN
      IF(IF(K).EQ.IF(J)) THEN                       IF(K)=1
      IF(K)=1                                       WRITE(6,500) J
      WRITE(6,21) J                                 J=5
      J=5                                        ENDIF
      ENDIF                                   ENDIF
      ENDIF                             30 CONTINUE
   43 CONTINUE                             STOP
      STOP                             500 FORMAT(1X,I2)
      END                                  END
```

Figure 2: A Subroutine 'tidied' by FLOPPY

2.6 FLOPPY on VAX/VMS

The format of the FLOPPY command on VMS is:

```
FLOPPY  filename  [/qualifiers]
```

Where 'filename' specifies the name of the input file of FORTRAN upon which the coding convention checks are to be made. You may use wild–cards in the filename; if more than one file is found matching the specification, then the files will be internally concatenated. Note that non–standard constructs such as INCLUDE statements will be treated as illegal statements by FLOPPY, and ignored. If you are using FLOPPY to tidy your Fortran (see /TIDY option), then 'filename' may be for instance an EDITF.DAT extracted with HISTORIAN option S, or likewise may be a file where the COMMON block declarations are hidden in INCLUDE statements. This will not jeopardize the indentation of DO loops and IF clauses, nor the re–numbering of statement labels.

2.6.1 VAX/VMS FLOPPY examples

```
$ FLOPPY myfile.for
```

Make all the standard coding convention checks on the FORTRAN file myfile.for.

```
$ FLOPPY/TREE/NOCHECKS myfile.for
```

Produce an output file for FLOW, and make no checks.

```
$ FLOPPY/IGNORE=(FRED,#MICHEL) myfile.for
```

Make all the available checks, but ignore the variable called FRED and the subroutine called MICHEL.

```
$ FLOPPY/CHECKS=(1,5,25,3)/FULL myfile.for
```

Check conventions 1,3,5 and 25, and list all lines from the source FORTRAN.

```
$ FLOPPY/CHECKS=(99,-1,-2,-20) my*.for
```

Check all conventions except numbers 1,2 and 20. Use all files beginning 'MY' and with filetype .FOR.

```
$ FLOPPY/NOLOG/OUT=output.lis  myfile.for
```

Send the FLOPPY output to a listing file, and disable the command parsing information.

```
$ FLOPPY/NOCHECKS/TIDY/INDENT=2 myfile.for
```

Produce a new FORTRAN file with all DO and IF clauses indented by two spaces. No coding convention checking is done. The new Fortran will be called FORTRAN.FOR.

```
$ FLOPPY/TIDY/STMNTS=(START=10,STEP=5)/FORTRAN=out.for myfile.for
```

Renumber statements starting at 10 (10, 15, 20 etc.) and write the new FORTRAN to the file OUT.FOR.

2.7 FLOPPY on IBM/CMS

Type "FLOPPY" to activate the program in full screen [2] mode, or "FLOPPY fn ft fm (options" to activate the program in line mode. The FLOPPY panel obtained by typing "FLOPPY" is shown in figure 3.

```
<=> FLOPPY VERSION 6.00 <=======> CODING CONVENTION CHECKER <====>

  Name of source Fortran file not yet given
                                          FN        FT      FM
                  Source  FORTRAN  ==>           FORTRAN   A
----------------------------------------------------------------------
a) Existing FLOPPY parameter file  ==>           FLOPIGN   A
b) Checks to be made          ==>  STANDARD
   Specify names to ignore    ==>  NO
----------------------------------------------------------------------
Generate a file for FLOW      ==>  NO
----------------------------------------------------------------------
Send FLOPPY Output to disk    ==>  YES
List all source FORTRAN lines  ==>  NO
----------------------------------------------------------------------
Tidy Fortran?  ==> NO      Output  ==>  OUTPUT   FORTRAN   A
 a) Adjust GOTOs to right of page  ==>  NO
 b) Indent DO/IF clauses           ==>  NO       Spaces  ==>  3
 c) Group FORMATs at routine ends  ==>  NO
 d) Renumber FORMAT statements     ==>  NO       Start   ==>  500
                                                 Step    ==>  10
 e) Renumber all other statements  ==>  NO       Start   ==>  10
                                                 Step    ==>  10

PF1: Help  (on  cursor)   PF2: Enter  a  CMS  Command     PF3: Exit
```

Figure 3: The FLOPPY panel

The default FLOPPY options are highlighted in the panel, and you may change these according to your requirements. By positioning the cursor on the item with which you want help, and pressing the PF1 key, FLOPPY will access the HELP file at the correct line. In addition, you may enter any CMS command whilst in the FLOPPY panel by first pressing the PF2 key. This is useful for example when you forget the whereabouts of your source Fortran file; press PF2 then enter "FILELIST * FORTRAN *". After execution of the command you will be asked if you want to return to the FLOPPY panel.

Alternatively, FLOPPY may be invoked in line–mode, in which case the format of the command is as shown in Figure 4.

[2] FLOPPY in full screen mode uses the IBM package IOS3270, a screen management facility for full–screen terminals, which is not standard REXX.

```
+--------+-----------------------------------------------------------+
|        |                                                           |
| FLOPPY | [ ? | fn [ft [fm]] [( Options ]]                          |
|        |                                                           |
|        | Options:                                                  |
|        |                                                           |
|        | [CHECKS [STANDARD|NONE|ALEPH|GALEPH|ONLINE|LIST|number_list]]|
|        | [DISK]                                                    |
|        | [FULL]                                                    |
|        | [GOTOS]                                                   |
|        | [GROUPF]                                                  |
|        | [IGNORE]                                                  |
|        | [INDENT    spaces]                                        |
|        | [OLD       fn ft fm]                                      |
|        | [OUTPUT    fn ft fm]                                      |
|        | [RENUMF    start_value[,step_value]]                      |
|        | [RENUMS    start_value[,step_value]]                      |
|        | [TIDY]                                                    |
|        | [TREE]                                                    |
+--------+-----------------------------------------------------------+
```

Figure 4: FLOPPY (IBM/CMS) command format

2.8 FLOPPY on Unix

Floppy operates on a single input file of Fortran code.

floppy [–l] [–c rules] [–f] [–o old] [–i names] [–j number] [–F] [–G]
[–r start,step] [–s start,step] [–n new] [–t flow] [file]

The meaning of each Unix qualifier is given in the list below.

♣ –l

The logging option causes Floppy to produce a verbose description of the selected options.

♣ –c

Defines the set of Coding Conventions to be checked. Specifying –cstandard causes the Standard set of checks to be made (see Section 2.2). Specifying –cn causes no checks to be made (useful for when code tidying options only are used). Specifying –ca causes all available checks to be made. Otherwise, a list of comma–separated numbers may be given. The number 99 has the special meaning: all rules, the number –99 means: no rules. So to check all rules except 1,3,5 and 31, use the qualifier –c99,–1,–3,–5,–31.

♣ –f

The full qualifier specifies that all source code lines should be listed by Floppy, rather than just those in breach of any of the specified coding conventions.

♣ –o

Use a previously generated file of rule numbers, ignore names, etc.. The tag value should be set to the name of the "old" file, normally the previous source Fortran name, appended with ".old".

♣ –i

Specify a list of variable or module names that are to be ignored by Floppy when checking coding conventions. The names should be specified as a comma–separated list. Module names should be preceded by a "#"sign. All names should be in upper case. Thus, to ignore subroutine GOOBAR, and all variables called FOOBAR, specify –i#GOOBAR,FOOBAR.

♣ –j

The indent option causes all DO loops and IF...THEN...ELSE...ENDIF clauses to be indented by the specified number of spaces. The default is 3 spaces. Values from 1 to 5 may be given.

♣ –F

Causes all FORMAT statements to be grouped at the end of each module.

♣ –G

Causes all GOTO clauses to be right–adjusted to column 72.

♣ –s

Specifies that all statement labels are to be renumbered. The start value, and optionally, the step value (default is 10) should be given. Thus to renumber starting at 100 and stepping by 20, use –s100,20.

♣ –r

Specifies that all FORMAT labels are to be renumbered. The start value, and optionally, the step value (default is 10) should be given. Thus to renumber starting at 1000 and stepping by 1, use –r1000,1.

♣ –n

Causes the "tidied" Fortran to be written to the specified file.

♣ –t

An output file is written for future processing by FLOW.

2.9 The meaning of the FLOPPY Qualifiers/Options

♣ *CHECKS (IBM/CMS,VAX/VMS) –C (UNIX)*

Define the coding convention checks to be made. If no list is given, then the checks marked by a '*' in Section 2.2 are made. If n=99 then all checks are made. If /NOCHECKS (VAX/VMS) or CHECKS NONE (IBM/CMS) is specified then no checks are made. If n is negative, then check number n is not made. Thus to make all the checks except numbers 3 and 31, specify /CHECKS=(99,–3,–31) on VMS, CHECKS 99,–3,–31 on IBM/CMS or –c99,–3,–31 on Unix.

♣ *DISK (IBM/CMS ONLY)*

Cause the output normally viewed at the terminal to be written to a file on disk. This file will have type FLOPLIS.

♣ *TREE (IBM/CMS,VAX/VMS) –T (UNIX)*

Cause a summary output file to be produced, containing a packed description of the source FORTRAN. The summary file contains such information as the list of all FORTRAN module names, their arguments, calling list, and so on. The file is unformatted; it should be used as input to the auxiliary tool called FLOW, and is unreadable at the terminal.

♣ *OUTPUT (IBM/CMS) –N (UNIX)*

Cause the reformatted FORTRAN output to be written on the filename specified. If no filename is given, then the output Fortran is written to a file called OUTPUT FORTRAN (IBM/CMS).

♣ *OUTPUT (VAX/VMS)*

Cause the output from FLOPPY (normally viewed at the terminal) to be sent to a disk file. If filename is not specified the output file will have the stem name of the source FORTRAN file, with a type of FLOPLIS .

♣ *FULL (IBM/CMS,VAX/VMS) –F (UNIX)*

Cause all source FORTRAN statements to be output, as opposed to only those breaking the specified coding conventions.

♣ *IGNORE (IBM/CMS,VAX/VMS) –I (UNIX)*

Specify a list of FORTRAN module and variable names to be ignored when the coding convention checks are made. Specify module names by preceeding the name with a # sign e.g. #MINUIT, specify variable names normally.

♣ *SPECIAL (IBM/CMS AND VAX/VMS ONLY)*

Specify that a special version of FLOPPY be used. The default 'special' version is STANDARD, which causes those checks marked by a '*' (see CHECKS) to be implemented.

```
STANDARD : Use the standard checks.
ALEPH    : Use the ALEPH standard checks.
GALEPH   : Variables beginning with G..... or xG.... are ignored.
```

Other special versions may be defined on request to the author.

♣ *LOG (VAX/VMS) –L (UNIX)*

Show a summary of the FLOPPY command parsing.

♣ *OLD (IBM/CMS,VAX/VMS) –O (UNIX)*

Each time FLOPPY is run, an "IGNORE" file is written with the user specifications for that particular run. If the OLD qualifier is used, one may specify an already existing "IGNORE" file. If the filename is omitted, then the filename used is obtained from the stem of the source FORTRAN file and the type FLOPIGN . Note that this qualifier does not affect the use of TIDY as the FORTRAN tidying parameters are not stored in the "IGNORE" file.

♣ *TIDY (IBM/CMS,VAX/VMS, DERIVED FOR UNIX)*

Write a new file of FORTRAN after re–formatting the input according to the qualifiers specified. The TIDY qualifier must be accompanied by at least one of the following qualifiers. If all you want to do is TIDY your Fortran, then use the /NOCHECKS or CHECKS NONE qualifier as well.

- ♣ *FORTRAN (VAX/VMS) (equivalent to OUTPUT,–n IBM/CMS,Unix)*
  Cause the reformatted FORTRAN output to be written on the filename specified. If no filename is given, then the output Fortran is written to a file called FORTRAN.FOR.
- ♣ *GOTOS (IBM/CMS,VAX/VMS) –G (Unix)*
  Right adjust all GOTO statements so that they finish in column 72.
- ♣ *INDENT (IBM/CMS,VAX/VMS) –j (Unix)*
  Indent DO and IF clauses by the specified number of spaces. The default is 3, and if specified, n should be in the range 1 to 5.

- ♣ *RENUMF (IBM/CMS) FORMAT VAX/VMS) –r (Unix)*
  Re–number FORMAT statements starting at n and stepping by m.
- ♣ *GROUPF (IBM/CMS,VAX/VMS) –F (Unix)*
  Group all FORMAT statements at the bottom of each module in which they appear.
- ♣ *STMNTS (IBM/CMS) RENUMS (VAX/VMS) –s (Unix)*
  Re–number all statements (not FORMATs) starting at n and stepping by m.

Chapter 3

Summary of Files produced by a FLOPPY run

Listed below are the files which will appear when you run FLOPPY. Some of these files will only appear if you select a particular FLOPPY option (shown in Status column).

Table 1: FLOPPY Files

VMS filename	IBM filename	Unix Filename	Description	Status
fn.FOR	fn FORTRAN	your.f	The source FORTRAN	Required
fn.FLOPFOR	OUTPUT FORTRAN	your.f.out	The ’tidied’ FORTRAN	Optional
fn.FLOPLIS	fn FLOPLIS	stdout	The FLOPPY output,	Optional
fn.FLOPIGN	fn FLOPIGN	your.f.old	The list of names to ignored by FLOPPY, and rule list	Optional
fn.FLOPTRE	fn FLOPTRE	your.f.floptre	The FLOPPY output to be input to FLOW	Optional
PROTRE.DAT	FILE PROTRE	protre.dat	The tree diagram from FLOW	Optional
PROCOM.DAT	FILE PROCOM	procom.dat	The table of COMMON block usage from FLOW	Optional
FLOW.PS	fn LISTPS	flow.ps	PostScript file for Structure Chart	Optional

Chapter 4

FLOW output

FLOW[3] uses a file produced by FLOPPY from your FORTRAN to analyse the flow of control between the modules, and produces a program control structure file together with a common block usage file. The output from the FLOW program is suitable for inclusion in a document such as a user’s guide.

It is also possible to produce a pseudo–structure–chart in PostScript format. Users are warned that, for charts containing more than about a dozen modules, the chart might be rather unreadable, as it is hard to optimize the module positions to minimize line crossings etc. whilst preserving the right call sequences. It is intended that this will be useful for "backward analysis", i.e. verifying that the SASD

3 FLOW is based on an original idea by P.Palazzi.

designed Structure Chart corresponds with the actual structure of the Fortran source. Please see Figure 5 and under /GRAPHICS below.

Figure 5: The FLOW Structure Chart for the FLOW program itself

```
This figure is available as a PostScript file called
FLOWFLOW.PS
```

```
********************             ProTre            ********************
                                 ======

                     Meaning of Symbols:
                     -------------------

                     .   ==> terminal node in the tree
                     *   ==> external procedure
                     >   ==> subtree node, expanded below
                     +   ==> multiply called terminal node
                     ]   ==> procedure calling only externals
-------------------------------------------------------------------------
                     ?   ==> module is in IF clause
                     (   ==> module is in DO loop

*************************************************************************


EXTERNAL procedure names will not appear


=============
Node name ==> FLOW
=============

 FLOW                                              :
     |-----PRODES                                  :Steer the FLOW program
               |-----INIARR   +                    :Initialise arrays
               |-----RDFLOP   >                    :Read the data from FLOPPY
               |-----EXTERN                        :Find names of external routines
               |         |-----RDFLOP   >          :Read the data from FLOPPY
               |?----PROTRE                        :Produce the FLOW diagram
               |         |-----SEARCH   +          :Finds the index for a routine name
               |         |-----LENOCC   ]          :
               |         |?----LENOCC   ]          :
               |         |?----LENOCC   ]          :
               |         |??---LENOCC   ]          :
               |         |??---LENOCC   ]          :
               |?----PROCHT                        :Produce the graphics SC
               |         |-----SEARCH   +          :Finds the index for a routine name
               |         |-----GRINIT              :Close the graphics package
               |         |         |-----LENOCC   ]:
               |         |-----CHTBOX   +          :Plots a box
               |         |-----SEARCH   +          :Finds the index for a routine name
               |         |((---CHTBOX   +          :Plots a box
               |         |((---GTX                 :Plots the text at a given positi
               |         |         |-----LENOCC   ]:
               |         |((---SEARCH   +          :Finds the index for a routine name
               |         |((---CHTLIN   ]          :Calculate and plot box intersect
               |         |(((--GRCLOSE             :Close the graphics package
               |         |         |-----LENOCC   ]:
               |?----PROCOM                        :Produce the COMMON block table
               |         |-----LENOCC   ]          :
               |         |-----LENOCC   ]          :
               |?----PROQRY                        :Interactively look at the tree
                         |-----CASCHG   >          :Convert any lower case to upper
                         |-----SEARCH   +          :Finds the index for a routine name
                         |-----CASCHG   >          :Convert any lower case to upper

=============
Node name ==> RDFLOP
=============

 RDFLOP                                            : Read the data from FLOPPY
     |-----TABENT                                  : Enter data into tables
               |((??-LENOCC   ]                    :
               |(----SEARCH   +                    : Finds the index for a routine name

=============
Node name ==> CASCHG
=============

 CASCHG                                            : Convert any lower case to upper
     |-----LENOCC   ]                              :
```

Figure 6: The FLOW diagram for the FLOW program itself. Subprograms that are called inside DO–loops are prefixed by one or more ('s, and those called conditionally are prefixed by one or more ?'s. In the FLOW shown, all external subprograms (e.g. UCOPY or SIN etc.) have been eliminated; this is an option.

```
 ********************              ProCom            ********************
                                   ======

                     Module names appear along x-axis
                     COMMON block names along y-axis

                     <Y>  ==> COMMON used in module
                     <N>  ==> COMMON not used (but is DECLARED)
                     < >  ==> COMMON not DECLARED
 *************************************************************************
1
           C C C F E G G G I L P P P P P R S T M A L N R M I
           A H H L X R R T N E R R R R R D E A I B E I E A N
           S T T O T I C X I N O O O O O F A B N S N N A X D
           C B L W E N L   A O C C D Q T L R E       T L   E
           H O I   R I O   R C H O E R R O C N             X
           G X N   N T S   R C T M S Y E P H T
          +----------------------------------------------------------------
 LUNIT    |  Y Y   Y Y   Y       Y   Y Y                                    |
 JOBCO1   |          Y Y       N   N                                        |
 JOBCOM   |          N                                                      |
 TABLE1   |                Y               Y                                |
 TABLES   |                Y                                                |
 FLOPP1   |                          N   Y   Y                              |
 FLOPPY   |                          N                                      |
 IGNORE   |                            Y                                    |
          +----------------------------------------------------------------
```

Figure 7: The COMMON block table for the FLOW program.

In the COMMON block table, the routine names appear along the top of the table, and the COMMON block names down the side. If the COMMON block is declared in a routine, and at least one variable used in that routine, then a "Y" appears in the corresponding row and column of the table. If a COMMON block is declared but not used in a routine, then an "N" appears.

4.1 FLOW on VAX/VMS

The FLOW command is invoked by:

```
FLOW  /qualifier  [/qualifiers]
```

Where the 'filename' is the file produced by FLOPPY when the /TREE qualifier is used.

4.2 Examples of VAX/VMS FLOW commands

Create a structure chart called STRUCTURE.CHT showing the calling sequence in myfile.FOR:

```
$ FLOW /INTREE=myfile.FLOPTRE /STRUCTURE=STRUCTURE.CHT
```

Write a DAF to be used in subsequent FLOW runs:

```
$ FLOW /INTREE=myfile.FLOPTRE /OUTDAF=myfile.DAF
```

Use a DAF and create a structure chart, ignoring all external procedure names (like VZERO, SIN, etc) and start the chart at subroutine INIJOB:

```
$ FLOW /INDAF=myfile.DAF /STRUC /NOEXT /NODE=INIJOB
```

Use the same DAF, and enter the interactive exploration facility of FLOW

```
$ FLOW /INDAF=myfile.DAF /QUERY
```

4.3 FLOW on CERNVM

Type "FLOW" to activate the program in full screen mode, or "FLOW (options" for line–mode. The FLOW panel is shown overleaf.

```
====> FLOW VERSION 2.00 <=========> Analyse Fortran Code <======>

 Fill in the blank field(s) as required.

 Input for FLOW
       Binary file from FLOPPY            ==>

 Interactively explore the calling tree  ==>

 Structure  Chart
        Node name at top of chart        ==>
         (Default is main program)
        File for text chart              ==>
        File for graphical chart         ==>
        Include EXTERNALs in chart ?     ==>

 COMMON block table file                 ==>

PF1: Help   PF2: Enter  a  CMS  Command     PF3: Exit
```

Figure 8: The FLOW panel

The format of the line–mode FLOW command is shown below:

```
+--------+---------------------------------------------------------+
| FLOW   | [ ? | fn [ ft [ fm]]]  [( Options ]]                     |
|        | Options:                                                |
|        | [QUERY]                                                 |
|        | [EXTERNALS]                                             |
|        | [NODE name]                                             |
|        | [STRUCTURE_CHART   fn [ft [fm]]]                        |
|        | [GRAPHICS          fn [ft [fm]]]                        |
|        | [COMMON_TABLE      fn [ft [fm]]]                        |
+--------+---------------------------------------------------------+
```

Chapter 5

The meaning of the FLOW Qualifiers/Options

The meaning of the options is described below:

♣ *INPUT FILE*

Specifies the name of a binary file produced using the TREE option in FLOPPY.

♣ *QUERY(IBM/CMS,VAX/VMS) –Q (UNIX)*

Enter the interactive exploration facility of FLOW ! You will be shown the full list of module names in the source, and can then specify one of them to see which modules it calls, and which modules call it. In this way you can move freely around the tree.

♣ *STRUCTURE_CHART(IBM/CMS,VAX/VMS) –S (UNIX)*

Specifies that a text file be written containing the calling tree of the source FORTRAN. The tree shows modules local to the code and external, and indicates whether they appear in IF or DO clauses. Stub modules (no calls), external modules (not local), multiply–called modules and sub–trees are all clearly indicated. If the FORTRAN source contained lines after each module declaration of the form:

```
C! This is a short comment
```

then these comments are shown against each module name in the tree.

♣ *GRAPHICS(IBM/CMS,VAX/VMS) –G (UNIX)*

Specifies that a PostScript file be written, which, when interpreted, will show the SASD–like structure chart of the source FORTRAN. It is recommended NOT to ask for EXTERNALS with this option. An attempt is made to optimize the positions of modules on the page, in order to minimise line crossings etc.. But this is often unsuccessful!

♣ *NODE (IBM/CMS,VAX/VMS) –N (UNIX)*

Specifies the top node in either the text or graphics structure chart. This is useful for restricting the output to a particular sub–tree of interest. The value nodename must be a PROGRAM, SUBROUTINE or FUNCTION name in the source FORTRAN.

♣ *NOEXTERNALS(IBM/CMS,VAX/VMS) –E (UNIX)*

Specifies that modules external to the source FORTRAN (e.g. SIN, COS, INDEX etc.) not be shown in the tree. The default is to show externals. This qualifier is to be used with the Query and Chart options.

♣ *COMMON_TABLE(IBM/CMS,VAX/VMS) –C (UNIX)*

Specifies that a table be written containing, on the y–axis the names of all COMMON blocks, and on the x–axis all the module names, contained in the source FORTRAN. At each place in the table a 'Y' indicates that the COMMON block appears in that module, and at least one variable from it is used in the module, a 'N' indicates that the COMMON block is declared but not used, and a blank indicates that the COMMON block is not declared in that module.

♣ *IGNORE (VAX/VMS) –I (UNIX)*

Specifies that the given list of module names is to be excluded from treatment when the Structure_Chart (–s) option is used.

Chapter 6

Problems and Installation

If you have problems using FLOPPY or FLOW, or if you discover bugs, then please contact the author:

- ♣ user JULIAN at CERNVM.CERN.CH, or
- ♣ user JULIAN at VXCERN.CERN.CH

If you want to install FLOPPY locally at your institution you should also contact the author, specifying which operating system you use.

Acknowledgments

I wish to thank Hans Grote for his help with implementing FLOPPY on top of FLOP. I also wish to thank Juergen Knobloch and Mike Metcalf for many helpful suggestions for improvements to the original version of FLOPPY.

I am indebted to Paolo Palazzi and Steve Fisher for their help and advice on the implementation of the original FLOW program.

Index