Program EZUP provides start up help in converting FORTRAN77 source codes to a form more closely resembling fortran 90 standards. EZUP was originally aimed at converting F77 programs to a form compatible with Lahey's Essential Lahey Fortran 90 (ELF90), but ELF90 has became a moving target and consistency with is no longer the primary aim of EZUP. EZUP is capable of doing a lot of the tedious chores found in conversions from F77 to F90, but conversion is usually not complete. EZUP makes ONLY SOME of the conversions and leaves most of the hard stuff to you. EZUP may also destroy the appearance of your code because the output file will not be "free form", and will look somewhat stiltedly like old-fashioned FORTRAN (rather than New fashioned fortran).

The input files must conform to old-fashioned column usage rules.

The current version number is 2.09 (21 Oct 2009) and seems to be functional under Microsoft Windows versions 9x/NT/XP, but has not been tried (at least not by me) with the Vista version (or, now “Windows 7”.) Successful executions have been made when version 2.09 has been compiled using Microsoft Fortran Power Station 4.0, Lahey LF90 and the GNU Cygwin version of g95 (use compiler option -O1 with g95). It was also tested successfully on a Linux system using the GNU gfortran compiler. The executable “ezup209.exe” included in the EZUP209.ZIP archive was built with the Microsoft FPS 4.o compiler. If that version does not run for you, compile it yourself using y our own f90 compiler and try it again.

```
*****************************************************************
*                                                               *
* DISCLAIMER: ALL THE USUAL FREEWARE DISCLAIMERS APPLY           *
*                                                               *
*             USE THIS PROGRAM AT YOUR OWN RISK!                *
*                                                               *
*             THE AUTHOR DISCLAIMS ANY LIABILITY FOR ANYTHING!  *
*                                                               *
*****************************************************************
```

---

Some rules about the input files:

1) Source code must be more or less standard F77 (or earlier) in form, and may not contain ANY blank lines.

2) Maximum number of do-levels is 20

3) Maximum number of labels in a program unit is 400, and the maximum number in a single instruction (including continuation cards) is 20.

4) Apostrophes may give trouble in some cases, such as

   IF (TMP(1).EQ.'Y') GO TO 400

   (but they may turn out ok, too).

5) Continuation card limit is ABOUT 20 (1450 characters total).

6) EZUP quits after 9999 lines (in a program unit, not the file (but it hasn't been tested on anything longer than about 3800 lines).

7) EZUP may choke on unlabeled DO loops, i.e. DO I=1,5 ... END DO, but versions 1.10+ try to avoid that.

8) Variable names (including any dimension info) should not exceed sixteen (16) characters.

---

About the Output files:

1) Sequence numbers in postions 73-80 will be lost.

2) Output is "old" FORTRAN formatted (although an occasional character or two may be pushed past column 72):

   a) Statement labels are in columns 1-5.
   b) Continuation lines all carry the & in column 6 and are broken for continuation at column 72 (with & in col. 73); un-indented statements begin in column 7.
   c) DO-LOOPS and IF-THEN-ELSE blocks may be indented.
   d) All C's in the first column are replaced by exclamation marks.
   e) Numeric labels are removed from DO nnnnn statements, and an "end do" statement is inserted to terminate each do-loop. (Unhappily, there will be 5 or 5 space codes in place of the deleted numerals.)

   f) Do-loops terminating on  nnnnn CONTINUE are rewritten as

```
   nnnnn MYCDEF=0

**********************************************************************
*************** A VERY, VERY IMPORTANT WARNING  ********************
**********************************************************************
*                                                                    *
*   Remark: The statements which terminate do-loops are retained     *
*   in the above dummy form because the CONTINUE might be the target *
*   of a "GO TO". EZUP does not check for that contingency and       *
*   is not smart enough to move nnnn labels to next executable       *
*   line. EZUP tries to put the "nnnnn MYCDEF=0" statement just      *
*   "outside" the appropriate "END DO". THAT MAY VERY WELL BE        *
*   THE WRONG PLACE! THE 'CONTINUE' MAY HAVE BELONGED "INDIDE"       *
*   THE "END DO".                                                    *
*                                                                    *
*   Exception: If the label nnnnn is never referenced, the whole     *
*   line will be deleted.                                            *
**********************************************************************
**********************************************************************
```

g) Other NNNNN CONTINUE statements become

```
   KKKKK IJKLMN=1
```

   Exception: If the label KKKKK is never referenced, the whole line will be deleted.

h) ENDIF is expanded to END IF, and ELSEIF is expanded to ELSE IF, or maybe they will remain as they were.

i) GOTO  may be expanded to GO TO

j) ALL statement labels are RENUMBERED consecutively, except that the renumbering is done before deleting unreferenced labels. So, some of the "new" labels will be missing, thereby fouling the just claimed consecutive feature. But, at least they will all ascenting numbers (in order of appearance).

k) "IMPLICIT NONE" is inserted immediately after the first line in each routine, and the line

```
    INTEGER :: IJLKMN, MYCDEF
```

    is inserted to define the variables added under items (h) and (g) above.

    Any existing IMPLICIT statements are deleted.

l) Things like:

```
       IF(X-XMAX)10,10,30
    10 IF(X-XMIN)30,20,20
    20 XRANG=0.0
    30 RETURN
```

  and

```
       GO TO (10,20,30,40,50), IVAR
```

are relabeled but no other changes are made..you must tackle them yourself if modification is needed.

m) Most characters will be converted to upper case (except in comments and formats).

n) A modest attempt is made to collect a list of variable names which appear immediately on the left of "=" signs. The list may be useful for assigning data types as required with the "implicit none" situation. Only strings of 16 or fewer characters are considered legal names, and the identification scheme very well may MISS ARRAY NAMES (as in VICTIM(J,K)=aval). The scheme may also fail to skip over things like RECL in statements like

Open(lunin,FILE='somefile', RECL=127,....).

Nor are any checks made to prevent the same string from appearing more than once. The source code was programmed to handle a maximum of 600 strings in a program unit.

The list is appended to the end of the actual code and is delimited by the exclamation mark in column/position no. 1.

o) General data typing effects (bugs and features)

```
REAL ABC, AVAR becomes:
REAL(4) :: ABC,AVAR
REAL*8 ABC, AVAR or DOUBLE PRECISION ABC, AVAR
becomes:       REAL(4) :: ABC,AVAR

REAL ABC, IVAR becomes:
   REAL(4) :: ABC
   INTEGER :: IVAR

COMPLEX Z1,ZZ,Z3 becomes:
   COMPLEX(4) :: Z1,Z2,Z3

COMPLEX*8 Z1,Z2,Z3,Z4 becomes
   COMPLEX(4) :: Z1,Z2,Z3,Z4

COMPLEX*16 Z1,Z2,Z3,Z4 becomes
   COMPLEX(8) :: Z1,Z2,Z3,Z4

CHARACTER A,B,C becomes
   CHARACTER(LEN=1) :: A,B,C

CHARACTER*32 ANAME, BNAME(10) becomes
   CHARACTER(LEN=32) :: ANAME, BNAME(10)

CHARACTER ANAME*32, BNAME*64 becomes:
   CHARACTER(LEN=32) :: ANAME
   CHARACTER(LEN=64) :: BNAME

CHARACTER*(*) STRING(*)  becomes
   CHARACTER(LEN=*) :: STRING(*)
  (I dunno whether that is right or wrong!)

DIMENSION A(12,34), MYARRY(20,200) becomes:
```

```
      REAL(4) :: A(12,34)
      INTEGER :: MYARRY(20,200)

    ****************************************************************
    ** NOTE ASSUMED DATA TYPES ARE FROM OLD-STYLE ("IMPLICIT ALL")**
    ** (A-H,0-Z) ARE REALS AND (I-N) ARE INTEGERS SCHEME.         **
    ****************************************************************

    LOGICAL ISTRUE, ISNOT becomes:
    LOGICAL :: ISTRUE, ISNOT
```

p) At the end of each program unit, the output file will contain a list of "apparent" variables. Unless the original code used the IMPLICIT NONE statement, you will probably need to create explicit data type definition statements for the variables found in this list. Doing that is a bit of a pain, but that may be the part of being frugal and not buying fancier compilers and conversion software.

   The variable name recognition scheme is primitive and looks only for strings like "      AONE=28.4E-16", from which it assumes that AONE represents a variable of unknown type. The scheme might also think that "READ(8,23,ERR=300)" means that ERR is also a variable. That usually will be detected, but MIGHT get through to the final list.

   The biggest failure of the scheme is the failure to recognize that ABCD(I,J)=.... makes ABCD a variable which will require a type declaration. That will probably not be necessary, however if there was a statement like DIMENSION ABCD(23,19) in the declarations portion of the code....

q) You will also see a (commented) list which shows which of the renumbered statement labels were deleted (sometimes the lines which contained them as well!). The relabeled code as it appeared before checking for unreferenced labels may (sometimes) be found in the file REVISED.TMP. In order to make this file accessible, it is NOT deleted automatically at the end of EZUP execution. However, it IS overwritten by each new program unit. So, if you want to see what labels and or lines were deleted in a particular routine, you will need to run it through EZUP on a "standalone" basis (a file of its own).

o) When EZUP knows that it cannot convert a particular statement, warnings will be displayed on the console and a commented similar statement will be placed in the output file.

p) Information regarding COMMON BLOCK processing will be found further donw in this file.

---

Things EZUP is definitely incapable of doing:

converting

1) FORMAT statements containing banned items such as 10X,1P,E10.3.
2) EQUIVALENCE statements.

3) BLOCK DATA routines.
4) PRINT * and READ * statements.

and

5) INTERFACE block construction (except as described below in version 1.13 update remarks).

CHARACTER statements may sometimes particularly convoluted in ways I have never imagined (or merely am too ignorant to comprehend) and may very well be converted to something worse than they were. Send along your pet-peeves.

---

How to use it:

Run EZUP from a command prompt just as with any other program. You will need to enter a numerical value (say 10) to be used as the new value for the lowest numbered statement label, and a value (say 10 again) for the increment between label values. Then a name for the input file, and a name for the output file. I think that path names are legal but only up to 32 characters and probably should not contain space codes.

If all works properly, you will be warned if you choose the name of an existing file as the name of the output file.

Several temporary files are created DURING EXECUTION OF ezup. The names of most of them will be found further down in this document. You may want to delete them if they are not deleted automatically but it is not necessary to do so.

You may occasionally see incomprehensible error messages. Do not expect the author to explain what they mean. He has probably forgotten. Send them to him anyway if you so desire.

The final output file will most surely NOT error or warning free, but it should be in a form which will save you a lot of grubby character by character editing which would otherwise be needed.

---

OTHER:

Contents of the EZUPxxx.zip file may be organized as individual zip files. The most recent version (2.09) contains:

- EZUP.PDF
- Sources.zip:
  - all.f90
  - rencom.f90
- Examples.zip:
  - atest.for
  - mytest.for
  - atest_b.out
  - mytest_b.out
  - intrfc00.f90
  - intrfc01.f90
  - module00.f90

```
            ezup.opt
            batchx.zip
      Execs.zip:
            ezup209.exe
```

wherein:

```
      EZUPxxx.EXE:  32 bit Microsoft Windows 95/NT/XP executable.
      ATEST.FOR:    Test file to check various operations; not
                    a compilable FORTRAN program.
      ATEST_B.OUT:   Result of EZUP on ATEST.FOR in batch execution mode.
      MODULE00.F90:  Sample MODULEXX.F90 file.
      INTRFC00.F90:  Sample INTRFCXX.F90 file.
      INTRFC01.F90:  Sample INTRFCXX.F90 file.
      EZUP.OPT:      Sample Batch-Mode Options file
      EZUP.PDF:      This file
      BATCHX.ZIP:    Sample data/output for a "batch mode" run
```

---

RECENT CHANGES:

19 July 2009:
Version 2.04 includes several source code changes. The most significant ones are in the handling of continuation "cards", and the method for trying to assign data type declarations to "COMMON" block variables when they are added to the MODULESxx.F90 file.

08 August 2009:
Version 2.05 will, I hope, do a better job of handling continuation cards and assignment of data type declarations within the MODULExx.F90 file. This version also does a somewhat better job of deleting source code lines involving type declarations for variables that also appear in MODULExx.F90 (the original common blocks). Version 2.05 also makes use of some temporary files not used in previous versions and does not delete them automatically because they are sometimes helpful in cleaning up compilation errors (mistakes made by EZUP) usually found in the first few compilation attempts with EZUP created output files. However.the temporary files will be overwritten by subsequent executions of EZUP.

02 Sept 2009:
In Version 2.07 a number of mostly irrelevant and debug comments have been deleted. In addition, to make the source codes more acceptable to the GNU g95 compiler, many statements like

```
       call sunx(phase,65,5)
```

have been expanded to forms

```
       ID1=65
       ID2=5
       call sunx(phase,ID1,ID2)
```

While the changes may cause reduced computational efficiency, they have simplified the job of insuring that all parameters passed between routines have the same size definitions.

21 October 2009:
Version 2.09 contains a number of fixups and corrections of which I haven't bothered to keep detailed records... trust me!

---

SOME CORRECTED AND/OR UNCORRECTED BUGS OF WHICH I AM AWARE

Because EZUP takes some shortcuts in attempting to determine statement types, labels and so forth, it is possible that certain kinds of statements will be misinterpreted and cause disastrous behavior of EZUP (it might even fill up your disk if you redirect console output to a file!). The problem is that when EZUP sees certain "phrases" it may assume that other things POSITIVELY, ABSOLUTELY MUST appear on the line. If they do not, then EZUP may find itself looking thorough the proverbial completely dark room seeking a black hat that is not there.

For example, the statement

```
      GWORD='GOTO'
```

causes EZUP to assume that it is dealing with a standard "GO TO" statement. If it IS, then there MUST be a "label" somewhere to the right of the second letter "O". But it looks in vain, and may possible conclude that the ASCII representation of the label is "     ", that is, five consecutive SPACE-codes. When EZUP later tries to match that target label with a list of statement labels found in columns 1 through 5, EZUP comes up quite confused!

In such a situation, EZUP's may issue a seemingly endless stream of error messages from subroutine SCRP. If you encounter such a state of affairs possible causes (but not the only ones) may be the presence of statements such as:

```
      GWORD='GOTO'
      KALL='CALL'
      ERRB=',ERR='
      ENDB=',END='
      READB='READ('
      WRITB='WRITE'
      DLP='DO'
      NDCARD='      END'
```

all of which are found in EZUP itself. The only way I have so far found to avoid trouble is to:

1) put a "C" in column 1 in the offending lines of the source code.
2) run EZUP
3) edit the output file to remove the !-symbols inserted in place of the C in column 1.
4) (perhaps) inserting type-declarations manually.

---

FEATURES and QUIRKS

1) Output files may contain some completely inappropriate statements:

```
! ** BEGIN  EZUP-GENERATED 'INTENT' STATEMENTS **
! ** END OF EZUP-GENERATED 'INTENT' STATEMENTS **
```

2) Output file INTRFCxx.F90 will contain the "guts" of the information needed for construction of INTERFACE for subroutines and functions in the original source code blocks. Users must do a bit of manual editing to copy the appropriate portions of that file into the proper location

of their actual program code, and (more work!) perhaps to adjust the INTENT statements to reflect proper IN,OUT or INOUT attributes for parameters.

However, there are SOME types of program units (BLOCK DATA, for example) for which meaningless INTERFACE blocks are created. Users will just have to deal with those on their own. Since such meaningless INTERFACE blocks are not automatically inserted into output program code, they should not cause serious problems.

3) Another undesirable quirk is that a SUBROUTINE involving no "passed arguments" will show up in toto in the GETL73 file. Just ignore it (if you can).

4) Automatic generation of INTENT statements for variables found in function and subroutine parameter lists. BUT, generally all vars are given generic INTENT(INOUT) property. The INOUT choice will usually be wrong, but the statements should be helpful in fixing associated errors generated by ELF90's petty pickiness.

5) EZUP some attempts to find data-type declarations ahead of the word FUNCTION for function declaration statements. As with much of EZUP, the outputs may be only approximately in a form which is completely acceptable to ELF90.

6) Several .TMP files may be left hanging around after execution ends. All may be deleted safely, but some may be useful in carrying out manual corrections to EZUP's converted output file.

7) Although EZUP handles "unlabeled" DO-LOOPS, but the resulting indentation is imperfect for last lines of DO-LOOPS which carried labels in the source code.

8) Type declaration statements found in the input F77 files may appear in the converted files in addition to EZUP generated type declarations picked out a function parameters.

---

COMMON BLOCK processing:

EZUP makes some modest attempts at trying to convert old fashioned "common blocks" to modules. Results apply only to a single source code file and there are a number of limitations (some removable, some not) as described elsewhere.

EZUP scans a source file (whose name you enter via keyboard) and attempts to locate all common blocks and declared variables. It constructs a partly converted source file, plus a special file named MODULExx.F90. Initially all variables found in the modules are declared to be of data-type UNKNOWN. The source file is then re-examined and attempts are made to replace the UNKNOWN with a more appropriate data-type if type declarations can be found. The process surely is not fool proof and some interactive editing will be required before you can convert MODULExx.F90 to a really legal " .F90 " file.

More details and GOTCHA's:

1) All variable and common block names are truncated to 16 characters.
2) Only ONE "blank" common block is allowed and it is always converted
   to MODULE DEFALT.
3) No checks are made to see that common blocks are of consistent
   size when declared in different program units.
4) Only the first declaration of a common block is processed.
5) An attempt is made to identify data-types for the variable names
   placed in the MODULExx.F90. The data-type used will be he first
   one found in the source file associated with each variable name.

   All detected common block variables which cannot be associated
   with a data-type assigned to a variable with the same name SOMEWHERE
   in the source file are given the default data type "UNKNOWN".
   That is, a variable named A3 will be declared in the MODULExx.F90
   file as "UNKNOWN :: xxxxxx" , where xxxxxx is the symbolic name
   of the variable.  YOU must deal with those yourself.

   NOTE WELL: There are some pitfalls in trying to associate variable
   names found in common block declarations with the data-type of
   variables names found in the source code. In particular, EZUP
   does not guarantee that the pair of names will come from the
   SAME PROGRAM UNIT, although it tries to impose that obvious
   requirement.

   For example, suppose that variables IX, IY and IZ AX appear
   in a statement of the form "COMMON /TRASH/ IX,IY,IZ" somewhere
   in the FIRST program unit in the source file, AND there is no
   corresponding data type assignment (expecting them to be
   treated implicitly as INTEGER):

   Now further suppose that some OTHER program unit of the source
   file (perhaps a subroutine) contains a variable declaration
   "REAL*4 IX,IY,IZ", but that the statement "COMMON /TRASH/ IX,IY,IZ"
   does NOT appear in that program unit.

   BAD MEDICINE! EZUP *MIGHT* assign the REAL(4) type to IX,IY and
   IZ in MODULE TRASH derived from the first appearance common
   block "TRASH"!

   MORALS: BE CAREFUL!

   DO NOT TRUST PAST SUCCESS AS A PREDICTOR OF THE FUTURE!

   IF TROUBLES OCCUR, PUT AN "IMPLICIT NONE" IN YOUR ORIGINAL
   SOURCE AND COMPILE IT TO GET A USEFUL LIST OF UNTYPED
   VARIABLES!

   DO NOT PLACE EXCESSIVE TRUST IN EZUP!

6) That is not the only problem: one of this EZUP version's failures
   is to REMOVE type declarations statements from the main source
   code when (if!) it duplicates them in the new MODULExx.F90 file.
   That will likely give a collection of compilation errors arising
   from multiple type declarations for each variable in a module.

   That might be a blessing in disguise, however, because it might
   warn you about possible "gotcha's" of type 5 (above). But, DON'T
   bet you job on it.

7) If your source code is broken into several files each of which
   is processed separately by EZUP be sure to rename the MODULExx.F90
   file before executing EZUP again with another source file. Every run
   of EZUP deletes the old MODULExx.F90 file and creates a new version
   based on the current input file. You will very likely need to do some
   juggling of the output MODULExx.F90 files to get everything synchronized.

8) Type declarations of variables that appear as SUBROUTINE or FUNCTION
   arguments are likely to appear at least twice within the final
   output file. Your first attempt to compile the output file will
   find those and you can the delete the surplus declaration statements.

```
**********************
*                    *
*     OTHER MORALS   *
*                    *
**********************
```

CAVEAT EMPTOR

AS ALWAYS, YOU USE EZUP AT YOUR OWN RISK!

If your old programs are even moderately complicated,
try to scrape up the money to buy a REAL (no pun!)
F90 compiler and forget about trying to convert your
old codes just to make use of a free compiler!

---

MISCELLANEOUS TOPICS

MISC-1. "BATCH MODE" Processing
EZUP 2.02 or later can now be executed in a "batch" mode.
By "batch" is meant that more than one source file can be
processed with a single execution of ezup itself. Separate
module and interface files are created for each source file.

The "batch" mode is entered if the working directory contains
a file named "EZUP.OPT". The file EZUP.OPT is organized as
a ser of "card images", with a "card image" being a single
line of text within a file.

A sample file looks like this:

```
    0   10   10    0    3    1    2
ATEST.FOR                            ATEST_B.OUT
MYTEST.FOR                           MYTEST_B.OUT
^----- name of input file -----^xxxxxxxx^-- name of output file here --^
```

ALL DATA POSITIONS SACRED! things must appear in certain "columns"
or else the whole execution will crash!!

If you use the EZUPOPT.OPT file to create an EZUP.OPT file, be sure to
delete the last line, namely:

```
^----- name of input file -----^xxxxxxxx^-- name of output file here --^
```

---

Here is some additional information on setting up an EZUP.OPT file:
The files in archive BATCHX.ZIP contain examples of "batch" runs of EZUP that process more than one input source (FOR or F77) file. Such calculations are controlled by the contents of the file EZUP.OPT (see below). If there is no EZUP.OPT file in the directory in which you placed ezupxxxx.exe then ezupxxx.exe operates in a "console input" mode and you must enter the requested information via your keyboard.

```
      OPTIONS FILE=EZUP.OPT:

      A FILE CARRYING RUN CONTROL INFO.

      LINE 1: FORMAT(7I5),  WITH INTEGER DATA:
              INTER, LFIRST, ISTEP, IKEEP, NDSPAC,KPTEMP,NFILES

      EXPLANATIONS:
        INTER    CONTROLS MODE OF OPERATION:
               = 0 MEANS NON-INTERACTIVE MODE; OPERATE WITH DATA IN THE
                   OPTIONS FILE OR INTERNAL DEFAULTS.

                   THIS MODE IS HANDIEST FOR BATCH AND BACKGROUND MODES
                   UNDER MS-DOS OR UNIX.

                   IF INTER=0 , THEN THE PROGRAM
                   WILL LOOK FOR A SECOND DATA CARD CARRYING THE SOURCE
                   AND OUTPUT FILE NAMES. IF IT ISN'T FOUND OR IF NAMES
                   ARE 'EMPTY', THE PROGRAM WILL STOP.

               = 1 MEANS THAT CONSOLE I/O IS TO BE USED FOR THESE DATA.
               = 2 MEANS THAT OPTIONS FILE WILL BE USED TO GET LABEL
                   VALUES, BUT CONSOLE WILL BE USED FOR ENTERING FILE
                   NAMES

        LFIRST = NUMERIC VALUE FOR FIRST LABEL OF EACH MODULE
                 ALLOWED RANGE IS 1 TO 10000; 100 IS SUGGESTED MAX.

        ISTEP  = NUMERIC VALUE OF INCREMENT OR STEP BETWEEN LABELS.
                 ALLOWED RANGE IS 1 TO 10000, BUT 100 IS SUGGESTED MAX.

        KEEP   = 0 MEANS THAT IT IS OK TO OVERWRITE AN EXISTING FILE
                   WHICH HAS THE NAME GIVEN FOR THE OUTPUT FILE.

               = 1 MEANS <<DO NOT>> OVERWRITE AN EXISTING OUTPUT FILE.

        NDSPAC = NUMBER OF COLUMNS TO BE USED FOR INDENTATION OF DO-LOOP
                 AND IF-THEN-ELSE BLOCKS. MAX IS 5, DEFAULT IS 2.

        KPTEMP = 0 MEANS DELETE TEMPCOPY.FOR AT END OF RUN
               = 1 MEANS DO NOT DELETE IT

        NFILES = NUMBER OF SEPARATE INPUT FILES TO BE RENUMBERED IN THE
                 INTER = 0 MODE.

 ---------------------------------------------------------------------

      LINES #2 through # NFILES+1: FORMAT(A32,8X,A32):

      FNAME1   = NAME OF INPUT (SOURCE PROGRAM) DATA FILE
      FNAME2   = NAME TO BE USED FOR OUTPUT (RENUMBERED) FILE
```

```
                FNAME1 <MUST> BEGIN IN COLUMN #1.
                FNAME2 <MUST> BEGIN IN COLUMN #32.

      NFILES  LINES OF THIS FORM ARE REQUIRED IF VARIABLE 'INTER'
      IS ZERO ON THE FIRST LINE (FIRST "CARD IMAGE").
```

---

For each input file there may be as many as three(3) output files:

1) The converted file, with whatever name you assign to it..
2) A file named "MODULExx.F90".
3) A file named "INTRFCxx.F90".

If several input files are specified by using an EZUP.OPT file then:

xx is 00 for the first processed file.
xx is 01 for the second processed file, and so forth.

Since only two digits are assigned, do not make NFILES greater than 99!

The MODULExx.F90 file is created only if there are COMMON BLOCKS in the source file.

```
************************* WARNING ** WARNING ************************
*                                                                   *
*         Any existing MODULExx.F90 and INTRFCxx.F90 files will be  *
*         replaced by the next execution of EZUP, so it is VERY     *
*         IMPORTANT to move or rename them before starting a run    *
*         with a new set of file names in EZUP.OPT                  *
*                                                                   *
*********************************************************************
```

MISCELLANEOUS FILES

TEMPORARY files created by EZUP are:

```
       TEMPCOPY.FOR
       INTRFACE.TMP
       TEMPSUBR.TMP
       SCRATCH1.TMP
       SCRATCH2.TMP
       PASS1.F90
       VARNAMES.TMP (list of detected variable names and types)
       COMMVARS.TMP (list of detected variable names and their types)
```

OUTPUT files are:

```
       MODULExx.F90   (may not be present if fewer than 3 lines long)
       INTRFCxx.F90   (may be irrelevant or empty in many cases)
       YOURFILE.xxx   (file name YOU chose for the outfile)
```

---

Another Moral: Free programs may or may not be worth every cent!

---

I welcome email containing any comments about both failures and successes with the use of EZUP. Please Send email to: w5ygr@aol.com. I will usually acknowledge receipt of such communications as soon as I can .. but I am not always in a place that allows daily email responses.

I hope that the program is useful to someone other than me, and I would like to hear from you if you find that it is useful to you too.

August Miller