Convert2FM.f95 is a program that tries to automate most of the conversion of a program to use the FM package for multiple precision computation.

To convert everything would require almost as much syntax analysis as a full compiler. Instead, the most commonly used Fortran features are handled, by using some shortcuts and simplifying assumptions about the syntax.

C2FM.INP is the input file -- the non multiple precision program.
C2FM.OUT is the output file -- the FM version.
Several other files are created for temporary information while Convert2FM runs, but these can be ignored.

Because not all the variables in the original program may need to be multiple precision, this conversion program uses the convention that the constant in the KIND part of the type declaration statement is a code telling when to convert a variable to multiple precision. Make a copy of the original program and call it C2FM.INP, then change each declaration there for which the variables will be multiple precision in the new version to one of the following three types. Use exactly these types, with all upper case letters and one blank.

```
REAL (KIND(3.1D1))     will be converted to  TYPE (FM),
COMPLEX (KIND(3.1D1))  will be converted to  TYPE (ZM),
INTEGER (KIND(31))     will be converted to  TYPE (IM).
```

For an existing program, decide which variables need to become multiple precision, and change each declaration to one of these. In the common case of a program that will not need multiple precision complex or integers, and all reals are double precision before the conversion, this can usually be done with one global change in an editor.

Any multiple precision functions defined in your program should be declared EXTERNAL along with the proper kind code. Omitting the external attribute might result in a compiler error message because of trying to initialize it.

If your program uses variables that will become multiple precision in a complicated way, then Convert2FM is likely to miss some changes. These will have to be found and changed by hand, but most of the time they will cause compiler error messages which show where the missed changes were. For example, if your program has multiple precision components of derived types or pointer aliases to multiple precision variables, Convert2FM might miss some changes.

The most time-consuming parts of conversion are usually changing constants,
```
   X = Y/3.7   should become   X = Y/TO_FM('3.7'),
```
changing type declarations,
```
   REAL (KIND(1.0D0)) :: X, Y   should become   TYPE (FM), SAVE :: X, Y
```
and inserting FM interface instructions like
```
   USE FMZM
   CALL FM_ENTER_USER_ROUTINE
   CALL FM_EXIT_USER_ROUTINE.
```
Convert2FM should be able to do these automatically.

Converting read/write statements is done by converting between machine precision and multiple precision, so that any associated formats do not have to be changed. These will often need to be changed by hand to get the desired multiple precision formats. This is especially true for TYPE (IM) numbers, since the machine precision integer overflow threshold is very low, so writing TO_INT(K) is seldom correct.

For example,

```
    WRITE (KW,120) N,H,TOL
```

becomes

```
    WRITE (KW,120) N,TO_DP(H),TO_DP(TOL)
```

after Convert2FM, which writes the multiple precision values H and TOL only to double precision accuracy.

Reads are similar but messier:

```
    READ (KR,*) (X(L),L=1,N)
```

becomes

```
    ALLOCATE( C2FM_TEMP(1:N) )
    READ (KR,*) (C2FM_TEMP(L),L=1,N)
    DO L = 1, N
       X(L) = C2FM_TEMP(L)
    ENDDO
    DEALLOCATE(C2FM_TEMP)
```

where X is type (fm) in the new version. C2FM_TEMP is a double precision variable in module C2FM_READS that Convert2FM puts at the start of the new program.