RC9000-10/RC8000

SW8585 Compiler Collection

XFORTRAN

A Preprocessor To RC FORTRAN

Keywords:
RC9000-10, RC8000, FORTRAN, Preprocessor, ISO FORTRAN, FORTRAN IV, RC FORTRAN

Abstract:
This manual describes XFORTRAN, a preprocessor used for converting FORTRAN IV source code to RC FORTRAN source code.

Date:
January 1989

PN: 991 11293

Table of Contents

1. Introduction

A preprocessor may be seen as a special kind of compiler, where the compilation is from one higher level language to another.

The preprocessor described in this paper transforms programs written in subset of FORTRAN IV to RC FORTRAN (ISO FORTRAN). The language constructs transformed are mainly concerned with input and output.

In more detail, the FORTRAN language accepted by the preprocessor is RC FORTRAN (as described in appendix A of the RC FORTRAN manual), extended with the language constructs described in the following sections.

The preprocessor is based on LR-parsing and syntax checks most of the program being processed, but only the language constructs not included in RC FORTRAN are transformed.

It should be noted that if too many (and severe) syntax errors are discovered it may happen that even if a construct should have been transformed, it will not be.

Furthermore it should be noted that only 72 characters may be used in a statement.

The preprocessor, in the following called XFORTRAN, should only be used if some of the language constructs described in the following are used. This is due to the fact that preprocessing is rather slow, XFORTRAN is able to process about 800 lines per minute in a process size of about 30 000 bytes.

2. The Language Constructs Transformed By XFORTRAN

The main difference between RC FORTRAN (a dialect of ISO FORTRAN) and the ANSI FORTRAN is the difference in use of input and output.

Standard FORTRAN (ANSI) uses a *unitnumber* as a reference to a file, whereas RC FORTRAN uses a *zone name* as a reference to a file (see chapter 5 in the RC FORTRAN Manual). The possibility to use direct access files does not exist, and some of the startements working on units are not allowed in the 'normal' way.

XFORTRAN tries to remedy part of these missing and/or changed facilities by accepting the following language constructs and changing them to equivalent RC FORTRAN statements:

```
DEFINE FILE
FIND
READ ( ' )
WRITE ( ' )
REWIND
ENDFILE
PAUSE
```

In the following the transformation of each of the above mentioned language constructs is described.

3. Sequential And Random Access To Files

As RC FORTRAN uses zone names instead of unitnumbers the preprocessor must change the unitnumbers in `DEFINE FILE`. It must be noted that unitnumbers, referring to files other than those used by `DEFINE FILE`, are not treated in this version of XFORTRAN.

The i/o system of RC FORTRAN does not support direct access files as allowed with `DEFINE FILE`, therefore all direct access to files must be simulated by some i/o routine capable to find the position of a random record in a file.

This version of XFORTRAN supports two different ways of using `DEFINE FILE`:

1 sequential
2 random access

The *sequential* mode is chosen if either `rand.no` or nothing is stated in the call of XFORTRAN (see section 5). The sequential use is cheap in both space and time in comparison to the random access.

The *random access* mode is chosen by setting `rand.yes` in the call of XFORTRAN, then the use of `DEFINE FILE` and the special `READ` and `WRITE` will work as though there is real random access to files. But it must be noted that it is very backing storage consuming because each record consists of a multiplum of segments (default value **one** segment).

In both cases the reading and writing must be performed *unformatted*, this is due to the fact that reading and writing cannot be executed with the same format because the first character of a line is used as a control character for vertical spacing (see section 5.4.5 in the RC FORTRAN Manual).

4. Statements

4.1 DEFINE FILE Statement

A file definition must obey the following syntax:

```
<direct access files> ::=

     DEFINE FILE <direct access file list>

<direct access file list> ::=

     <direct access file list><direct access file>
     |  <direct access file>

<direct access file> ::=

     <unit> ( <noofrec>, <maxsize>, U, <ass.var> )

     <unit>         ::=  unsigned integer between 1 and
                         99

     <noofrec>      ::=  unsigned integer

     <maxsize>      ::=  unsigned integer

     <ass.var>      ::=  variable with as most 6
                         characters, if it is longer
                         the exceeding characters are
                         cut off in subsequent use.
```

The `define file` statement is a declaration and must therefore obey the same order in a program unit as other declarations (see chapter 6.1 in the RC FORTRAN Manual).

Processing this statement XFORTRAN will define a zone with the name `FIL` concatenated with the unitnumber, the associated variable will be declared as an integer. Furthermore the logical function name `setposition` is declared.

The recordsize <maxsize> should be used with care in case of rand.yes. <maxsize> is the maximum number of integers which can be contained in one record. As already mentioned the record size in XFORTRAN will be a multiplum of segments which is calculated as follows:

```
record size = <maxsize> * 2 / 512 + 1
```

Example

Assume that XFORTRAN meets:

```
.
.
DEFINE FILE 5 (100, 7, U, COUNT)
. .
```

This will be transformed to:

```
.
.
DEFINE FILE 5 (100, 7, U, COUNT)
ZONE FIL5 (128, 1, STDERROR)
INTEGER COUNT
LOGICAL SETPOSITION
. .
```

And as the first executable statements of the program:

```
CALL OPEN (FIL5, 'FIL5', 0)
COUNT = 1
```

For the explanation of setposition and open the reader is referred to chapter 5 in the RC FORTRAN Manual.

It should be noted that only backing storage files (i.e. disk files) can be defined this way. If the file refers to a magnetic tape a runtime error will appear, this is due to the second parameter in the open call.

Furthermore is should be noted that files used in a program apart from in and out must be declared in the directives to the operating system (see the examples in section 6).

4.2 FIND Statement

The find statement obeys the following syntax:

```
<find statement> ::=

     FIND (<unit>'<integer expression>)
```

The statement should cause the next input record to be found while the present record is being processed, thereby increasing the speed of the program. This statement has no effect in RC FORTRAN and will be transformed to a comment.

4.3 Direct Access READ Statement

The `direct access read` statement obeys the following syntax:

```
<direct access read> ::=

     READ (<unit>'<integer expression>) <i/o-list>
```

The next record is read into the `<i/o-list>` from the file `FIL<unit>` and the associated variable is assigned the value `1+<integer expression>`. In the sequential case the `<integer expression>` has no effect on the record chosen for input. A label should not prefix such a statement.

Example

```
      PROGRAM JOB1
      .
      .
      DEFINE FILE 5 (100, 7, U, COUNT)
      .
      .
      .
      READ (5 ' COUNT +2) A, B
```

In the sequential case this will be transformed to:

```
      PROGRAM JOB1
      .
      .
C*    DEFINE FILE 5(100, 7, U, COUNT)
      ZONE FIL5(128, 1, STDERROR)
      INTEGER COUNT
      LOGICAL SETPOSITION
      .
      .
      .
      CALL OPEN (FIL5, 4, 'FIL5', 0)
      COUNT = 1
      .
      .
      .
      READ (FIL5) A, B
      COUNT = 1 + COUNT +2
C*    READ (5' COUNT +2) A, B
```

In the random access case:

```
      PROGRAM JOB1
      .
      .
C*    DEFINE FILE 5(100, 7, U, COUNT)
      ZONE FIL5(128, 1, STDERROR)
      INTEGER COUNT
```

```
      LOGICAL SETPOSITION
      .
      .
      .
      CALL OPEN (FIL5, 4, 'FIL5', 0)
      COUNT = 1
      .
      .
      .
      CALL SETPOSITION (FIL5, 0, (COUNT +2 -1)*1)
      READ (FIL5) A, B
      COUNT = 1 + COUNT +2
C*    READ (5' COUNT +2) A, B
```

4.4 Direct Access WRITE Statement

The `direct access write` statement obeys the following syntax:

`<direct access write> ::=`

`WRITE (<unit> ' <integer expression>) <i/o-list>`

The `<i/o-list>` is written as the next record in the file `FIL<unit>` and the associated variable is assigned the value `1+<integer expression>`.

As with `read` the `<integer expression>` has no effect on the record chosen for output in case of sequential use. A label should not prefix such a statement.

4.5 REWIND Statement

The syntax is:

`<simple statement> ::= REWIND <unit>`

A `unit` (file) declared by a `DEFINE FILE` statement is rewound, so that a subsequent `read` or `write` will refer to the first record of the file.

Example

```
      .
      .
      REWIND 5
      .
      .
```

This will be transformed to:

```
      .
      .
C*    REWIND 5
      CALL SETPOSITION (FIL5, 0, 0)
```

4.6 ENDFILE Statement

The syntax is:

```
<endfile statement> ::= ENDFILE <unit>
                     /  END FILE <unit>
```

A unit (file) declared by a DEFINE FILE statement is closed and released from the running program. It should be noted that if a file is going to be used in subsequent programs an endfile statement should be used as the last action on the file before termination.

Example

```
      .
      .
      ENDFILE 5
      .
      .
```

This will be transformed to:

```
      .
      .
C*    ENDFILE 5
      CALL CLOSE (FIL5, .TRUE.)
      .
      .
```

4.7 PAUSE Statement

The syntax is:

```
<pause statement> ::= PAUSE [ <integer> ]
```

The statement should display an integer on the operator console and the program should stop until the operator causes the program to resume execution. In the actual version, the statement is legal, but blind (i.e., transformed to a comment).

5. The Function Of The Preprocessor

The preprocessor will be a backing storage file (disk) called XFORTRAN.

The of XFORTRAN is exactly as the call of the FORTRAN compiler, with the following exceptions:

1 The source file, if any, must be the **first** parameter of the call,

2 only **one** source file is allowed.

To summarize, the call of XFORTRAN follows the following syntax:

`[<bs file>=] xfortran [<source file>] {<modifier>}0-*`

For further explanation see Appendix B in the RC FORTRAN Manual.

The `modifier` list has been extended with two new modifiers which only can be used in the call of XFORTRAN - these are:

1 `rand. {yes/no}`, which already has been explained. The default value is `rand.no`.

2 `xref. {yes/no}`. which makes it possible to get a cross reference of a program. This is explained further in Appendix A. The default value is `xref.no`.

XFORTRAN outputs the transformed program on a workfile. This workfile will replace the original source file in the call of the compiler. The workfile is created at each call of XFORTRAN and the name of the file will be unique within the user base (the name will be `wrk` concatenated with six digits). After preprocessing the FORTRAN compiler is called with the workfile as source file and all other parameters unchanged (except for the two modifiers just mentioned).

```
fortran program --> XFORTRAN
                       |
                       |
                    transformed
                    program on
                    wrk <integer> --> FORTRAN --> object code
                                         |
                                         |
                                      listing of
                                      transformed
                                      program results
                                      etc.
```

If the user wants to preserve the work file with the transformed program after the preprocessing this is possible by running the job with the job options `preserve yes` (see the BOSS User Manual). The name of the work file can be found by inserting a `search temp` in the job. This utility call will list the temporary file which are visible for the user, among these will be the workfile (for further explanation of utility programs, see the Utility Program Manuals, part 1 and 2).

Files used in FORTRAN programs must either be 'declared' by a utility program or by using the monitor procedures which are available (see the Monitor and Algol 8 manuals).

If the file does not exist as a catalog entry visible from the user base a runtime error will appear whjen the file is opened for communication with the program.

Example

Assume that a FORTRAN program uses the files identified by unit 3 and 5, then the following commands can be used:

```
job xyz size 30000
(fil3 = set 21 disc2
 fil5 = set 21 disc2
.
.
binprg = xfortran list.yes
.
.
finis)

PROGRAM TEST
DEFINE FILE 3 (200, 3, U, C1)
DEFINE FILE 5 (100, 3, U, C2)
.
.
.
```

5.1 Error Messages From XFORTRAN

The errors reported from XFORTRAN in the current version concern limitations in table sizes only (with one exception). The errors will be reported as a written text on current output and XFORTRAN will stop processing, hence the program will not be compiled by the FORTRAN compiler.

The following messages may appear:

parse stack overflow (stackmax)
The stack used in the syntax checking is too small, the limit `stackmax` must be changed.

end of file encountered
The program being preprocessed is exhausted, probably because the program contains too many syntax errors.

too many file definitions (filemax)
The program contains too many `DEFINE FILE` statements, `filemax` must be changed.

parameter to the preprocessor too small (fpmax)
The list used to contain the call of the compiler cannot contain the whole parameterlist, `fpmax` must be changed.

6. Examples

The following shows an example of the use of XFORTRAN in two versions. The program is listed in the original version and in the preprocessed version.

The example show the reading and writing of a file defined by a `DEFINE FILE` statement. Both a sequential and a rabndom access run is shown.

Source file

```
JOB XYZ 3 1 SIZE 30000 TIME 35
(MODE LIST.YES
 FIL5=SET 42
 HEAD CPU
 BINPRG=XFORTRAN LIST.YES RAND.YES
 HEAD CPU
 BINPRG
 HEAD CPU
 FINIS)
       PROGRAM TESTJOB5
       DEFINE FILE 5(20,5,U,COUNT)
       DIMENSION A(5), B(10), C(15)
       ZONE OUT; EXTERNAL OUT
       INTEGER J, K, L
       COMMON /AB/ A,B
       DATA A/7,7,7,7,7/
       DATA B/6,6,6,6,6,6,6,6,6,6/
C TEST OF DEFINE FILE


       DO 1 I=1,5
       WRITE(5'I) A(I)
1      CONTINUE

       DO 2 J=1,10
       WRITE(5'J) B(J)
2      CONTINUE

       REWIND 5 DO 3 K=1,15
       READ(5'K) C(K)
3      CONTINUE
```

```
        DO 4 K=1,15
        WRITE(OUT,5) C(K)
5       FORMAT (F5.0)
4       CONTINUE
        END
```

The result of the sequential run (the parameter `rand.yes` has been omitted):

```
*FIL5=SET 42
*HEAD CPU
XYZ3  1988.07.20  12.12.13  CPU: 0.10 SEC.
*BINPRG=XFORTRAN LIST.YES RAND.NO
*BINPRG=FORTRAN WRK000035 LIST.YES
         1      PROGRAM TESTJOB5
         2 C*   DEFINE FILE 5(20,5,U,COUNT)
         3      ZONE FIL5( 128, 1, STDERROR)
         4      INTEGER COUNT
         5      LOGICAL SETPOSITION
         6      DIMENSION A(5), B(10), C(15)
         7      ZONE OUT; EXTERNAL OUT
         8      INTEGER J, K, L
         9      COMMON /AB/, A,B
        10      DATA A/7,7,7,7,7/
        11 C TEST OF DEFINE FILE
        12      DATA B/6,6,6,6,6,6,6,6,6,6/
        13
        13      CALL OPEN(FIL5  , 4,  'FIL5', 0)
        14      COUNT  = 1
        15      DO 1 I=1,5
        16      WRITE(FIL5) A(I)
        17      COUNT  = 1 + I
        18 C*   WRITE(5'I) A(I)
        19 1    CONTINUE
        20
        20      DO 2 J=1,10
        21      WRITE(FIL5) B(J)
        22      COUNT  = 1 + J
        23 C*   WRITE(5'J) B(J)
        24 2    CONTINUE
        25
        25 C*   REWIND 5
        26      CALL SETPOSITION(FIL5, 0, 0)
        27      DO 3 K=1,15
        28      READ(FIL5) C(K)
        29      COUNT  = 1 + K
        30 C*   READ(5'K) C(K)
        31 3    CONTINUE
        32
        32      DO 4 K=1,15
        33      WRITE(OUT,5) C(K)
        34 5    FORMAT(F5.0)
        35 4    CONTINUE
        36      END
FIN.  END
```

The result of the sequential run (notice that all date items are valid):

```
*HEAD CPU
XYZ3  1988.07.20  12.12.25  CPU: 3.73 SEC.
*BINPRG

     7
     7
     7
     7
     7
     6
     6
     6
     6
     6
     6
     6
     6
     6
     6
END
*HEAD CPU
XYZ3  1988.07.20  12.12.26  CPU: 3.93 SEC.
*FINIS

END  13 SEC  JOB XYZ3  LOG XYZ  DATE 1988.07.20  12.12.17
```

The second run shows the random access (the parameter `rand.yes` has been used):

```
*FIL5=SET 42
*HEAD CPU
XYZ3  1988.07.21  12.10.10  CPU: 0.10 SEC.
*BINPRG=XFORTRAN LIST.YES RAND.YES
*BINPRG=FORTRAN WRK000015 LIST.YES
 1       PROGRAM TESTJOB5
 2 C*    DEFINE FILE 5(20,5,U,COUNT)
 3       ZONE FIL5 ( 128, 1, STDERROR)
 4       INTEGER COUNT
 5       LOGICAL SETPOSITION
 6       DIMENSION A(5), B(10), C(15)
 7       ZONE OUT; EXTERNAL OUT
 8       INTEGER J,K,L
 9       COMMON /AB/ A,B
10       DATA A/7,7,7,7,7/
11 C TEST OF DEFINE FILE
12       DATA /76,6,6,6,6,6,6,6,6,6/
13
13       CALL OPEN(FIL5  , 4, 'FIL5', 0)
14       COUNT = 1
15       DO 1 I=1,5
16       CALL SETPOSITION(FIL5 , 0, (I - 1)* 1)
17       WRITE(FIL5) A(I)
18       COUNT = 1 + I
```

```
C*    WRITE(5'I) A(I)
1     CONTINUE

      DO 2 J=1,10
      CALL SETPOSITION(FIL5 , 0, (J - 1)* 1)
      WRITE(FIL5) B(J)
      COUNT = 1 + J
C*    WRITE(5'J) B(J)
2     CONTINUE

C*    REWIND 5
      CALL SETPOSITION(FIL5, 0, 0)
      DO 3 K=1,15
      CALL SETPOSITION(FIL5, 0, 0)
      READ(FIL5) C(K)
      COUNT = 1 + K
C*    READ(5'K) C(K)
3     CONTINUE

      DO 4 K=1,15
      WRITE (OUT,5) C(K)
5     FORMAT (F5.0)
4     CONTINUE
      END

FIN.  END
```

The result of the run (notice that the five last date items are undefined):

```
*HEAD CPU
XYZ3  1988.07.21  12.12.21  CPU: 3.97 SEC.
*BINPRG

         6
         6
         6
         6
         6
         6
         6
         6
         6
         6
         0
         0
      E+10
         0
         0
END
*HEAD CPU
XYZ3  1988.07.21  12.12.23  CPU: 4.22 SEC.
*FINIS


END  13 SEC  JOB XYZ3  LOG XYZ  DATE 1988.07.21  12.12.25
```

Appendix A. Crossreference

The possibility to get a program crossreferenced exists in XFORTRAN. The crossreference is activated by the parameter `xref.yes` in the call of the preprocessor. The default value in `xref.no`.

The crossreference sorts one programunit (subroutine, function and mainprogram) at a time and the identifiers are listed in alphabetical order along with the linenumbers where the identifiers appear.

Identifiers created by XFORTRAN (such as zonenames) are not listed. The maximum of significant characters in an identifier is determined by XFORTRAN (in the current version 12).

The following messages appear from the crossreference program:

identifier appears too many times
The constant `noofappear` is too small.

binary tree too small
The sorttree is too small, `bintreelimit` msut be changed.

The following shows an example of the use of `xref.yes`.

The source program:

```
JOB XYZ 4 1 TIME 35 SIZE 30000
(MODE LIST.YES
 HEAD CPU
 PRG=XFORTRAN LIST.YES XREF.YES
 HEAD CPU
 PRG
 HEAD CPU
 FINIS)
                    SUBROUTINE SUM(P1,P2)
                    INTEGER P1,P2,SUM1
                    COMMON /ALL/ SUM1
                    SUM1=P1+P2
                    END
```

```
      SUBROUTINE DIFF(D1,D2)
      INTEGER D1,D2,SUM1
      COMMON /ALL/ SUM1
      SUM1=D1-D2
      END

      PROGRAM EXAMPLE
      INTEGER SUM1
      ZONE OUT; EXTERNAL OUT
      COMMON /ALL/ SUM1
      DO 50 I=1,20
         CALL SUM(I,I+1)
         CALL DIFF(I,1)
50 CONTINUE
      WRITE(OUT,60) SUM1
60 FORMAT(///,' SUM = ' ,I5)
      END
```

The result of the run:

```
*HEAD CPU
XYZ  1988.07.22  12.12.30  CPU: 0.08 SEC.
*PRG=XFORTRAN LIST.YES XREF.YES
*PRG=FORTRAN WRK000054 LIST.YES
       1          SUBROUTINE SUM(P1,P2)
       2          INTEGER P1,P2,SUM1
       3          COMMON /ALL/ SUM1
       4          SUM1=P1+P2
       5          END
       6
       6          SUBROUTINE DIFF(D1,D2)
       7          INTEGER D1,D2,SUM1
       8          COMMON /ALL/ SUM1
       9          SUM1=D1-D2
      10          END
      11
      11          PROGRAM EXAMPLE
      12          INTEGER SUM1
      13          ZONE OUT; EXTERNAL OUT
      14          COMMON /ALL/ SUM1
      15          DO 50 I=1,20
      16             CALL SUM(I,I+1)
      17             CALL DIFF(I,1)
      18  50      CONTINUE
      19          WRITE(OUT,60) SUM1
      20  60      FORMAT(///,' SUM = ',I5)
      21          END

FIN.  END
*CROSSREF WRK000055



CROSSREFERENCE SUM
```

```
        ALL 3
        P1 1 2 4
        P2 1 2 4
        SUM 1
        SUM1 2 3 4



        CROSSREFERENCE DIFF

        ALL 8
        D1 6 7 9
        D2 6 7 9
        DIFF 6
        SUM1 7 8 9



        CROSSREFERENCE MAIN

        ALL 14
        DIFF 17
        EXAMPLE 11
        I 15 16 16 17
        OUT 13 13 19
        SUM 16
        SUM1 12 14 19
END  28
*HEAD CPU
XYZ4  1988.07.22  12.12.46  CPU: 4.53 SEC.
*PRG


SUM =     19
END
*HEAD CPU
XYZ4  1988.22.07  12.12.47  CPU: 4.61 SEC.
*FINIS


END  16 SEC  JOB XYZ4  LOG XYZ  DATE 1988.07.22  12.12.50
```