Universiteit Groningen – Kapteyn Laboratorium

SHELTRAN

Contents

Chapter 1

Introduction.

The program SHLTRN accepts a program written in the SHELTRAN language and generates the corresponding FORTRAN 77 program and, optionally, a source listing.

SHELTRAN is 'structured FORTRAN' - this being ANSI FORTRAN 77 with certain facilities replaced by corresponding 'structured' facilities, plus additional facilities with no FORTRAN equivalents.

SHELTRAN declares statement labels (except for the FORMAT statement) illegal, and hence any control statement requiring statement labels as arguments is also illegal. These statements are replaced by closely-matching 'structured' facilities with controlled exits. SHELTRAN also provides various forms of the repetition construct (allowing controlled exits).

Chapter 2

Description of the language.

In describing the SHELTRAN language the following nomenclature is used:

- SHELTRAN keywords appear in capitals. (SHELTRAN keywords always start on a new line and must be followed by at least one blank)
- Syntactic elements are enclosed within < >.
- Items enclosed within [ ] are optional.
- <statement> is used to denote any SHELTRAN and/or FORTRAN statement(s).

2.1 Statement labels.

In SHELTRAN the statement label is forbidden for all executable statements. The only context in which the statement label is permitted is in combination with a FORMAT statement. In this case the label must be less than 10000.

As a direct consequence the following normal FORTRAN statements become unuseable in SHELTRAN:

- ASSIGN
- all forms of GOTO
- all forms of IF
- DO
- the RETURN <variable> form of the RETURN statement
- the use of statement numbers in END= or ERR= clauses in I/O statements

- the use of statement numbers as arguments of CALL statements

To replace FORTRAN control structures several new SHELTRAN structures have been introduced.

2.2 IF construct.

2.2.1 Form:

```
 IF <logical expression>
 THEN
   <statement>...
[ ELSEIF <logical expression>
 THEN
   <statement>...    ]
[ ELSE
   <statement>...    ]
 CIF
```

2.2.2 Notes:

- <logical expression> represents any valid FORTRAN logical expression which can take a value .TRUE. or .FALSE. It need not be surrounded by parentheses.
- ELSEIFs when present must precede any corresponding ELSE.
- The ELSE clause is optional.
- Instead of CIF ENDIF may also be used.

2.2.3 Actions:

When the IF or ELSEIF condition has the value .TRUE., the statements following are executed and control is transferred to the statement following the CIF.

When the value is .FALSE., control is passed to the next ELSEIF and this <logical expression> is evaluated.

When the IF and all ELSEIF clauses have a value which is .FALSE., control is passed to the statement following the corresponding ELSE or CIF.

2.2.4 Examples:

```
 IF KEY.EQ.K1
 THEN
```

```
  KODE=1
ELSEIF KEY.EQ.K2
THEN
  KODE=2
ELSE
  KODE=-1
CIF
```

```
IF X.LT.0
THEN
  X=999.0
CIF
```

2.3 SELECT construct.

2.3.1 Form:

```
 SELECT <arithmetic expression>
 CASE n1 [,n2...]
[   <statement>...      ]
[ CASE m1 [,m2...]
   <statement>...      ]
 OTHER
[   <statement>...      ]
 CSELECT
```

2.3.2 Notes:

- n1 [,n2...] represents a series of unsigned integer values, separated by commas. At least one integer must be present for each CASE statement.
- the integer values may appear in any order in the CASE clause.
- the CASE clauses need not contain all values in the range 1 to N, where N is the largest integer value used. However, a specific value may only appear once.
- the largest value used in a SELECT construct must not exceed 198.
- the OTHER clause is mandatory.

2.3.3 Actions:

The expression after the SELECT is evaluated and truncated to an integer value. Control is transferred to the statements following the CASE statement which contains the value of the SELECT expression. If no such value can be found, the OTHER clause is selected. Finally control is transferred to the statement after CSELECT.

2.3.4 Example:

```
SELECT COLOR+1.0
CASE 1
  COLTXT='BLACK'
CASE 2
  COLTXT='RED'
CASE 3
  COLTXT='GREEN'
CASE 5
  COLTXT='BLUE'
CASE 8
  COLTXT='WHITE'
OTHER
  COLTXT='MIXTURE'
CSELECT
```

2.4 WHILE construct.

2.4.1 Form:

```
 WHILE <logical expression>
   <statement>...
[   XWHILE
   <statement>...     ]
 CWHILE
```

2.4.2 Actions:

Whilst <logical expression> has a value .TRUE., the statements within the loop WHILE-CWHILE will be executed, and <logical expression> re-evaluated. When <logical expression> has a value .FALSE., control is transferred to the statement following CWHILE. At any point in the WHILE loop an XWHILE statement may be inserted. This statement transfers control to the statement following CWHILE independently of the value of <logical expression>.

2.4.3 Example:

```
WHILE I.LE.72
```

```
    CHR=LINE(I)
    IF CHR.EQ.BLANK
    THEN
      I=I+1
    ELSE
      IF CHR.GE.'A'.AND.CHR.LE.'Z'
      THEN
        XWHILE
      CIF
    CIF
  CWHILE
```

2.5 REPEAT construct.

2.5.1 Form:

```
  REPEAT
    <statement>...
[   XREPEAT
    <statement>... ]
  UNTIL <logical expression>
```

2.5.2 Actions:

The statements between REPEAT and UNTIL are executed, and <logical expression> evaluated. When <logical expression> has the value .TRUE., control is transferred to the the statement after UNTIL; if it is .FALSE., the statement after REPEAT gains control. At any point in the REPEAT loop an XREPEAT statement may be inserted. This statement transfers control to the statement following UNTIL, independently of <logical expression>.

2.5.3 Example:

```
REPEAT
  CALL WAIT(EVENT)
  SELECT EVENT
  CASE 1
    CALL TYPEIN
  CASE 2,3,7,4
    CALL SERVE
```

```
  OTHER
    FINISH=.TRUE.
  CSELECT
UNTIL FINISH
```

2.6 FOR construct.

2.6.1 Form:

```
 FOR <loop control>
   <statement>...
[   XFOR
   <statement>... ]
 CFOR
```

2.6.2 Note:

The rules for writing <loop control> are identical to those for the FORTRAN DO-loop.

2.6.3 Actions:

The actions follow the FORTRAN DO-loop control actions. At any point in the FOR-loop an XFOR statement may be inserted. This statement transfers control to the statement following CFOR.

2.6.4 Example:

```
 FOR I=1,80
   LINE(I)=' '
 CFOR
```

2.7 PERFORM and PROC.

2.7.1 Form:

```
C    To invoke a procedure section:
     PERFORM <procedure name>
```

```
C    To define a procedure section:
     PROC <procedure name>
       <statement>...
     CPROC
```

2.7.2 Notes:

- The procedure name must not be longer than 31 characters.
- The statements in a PROC may not include:
  - STOP or RETURN
  - XWHILE, XREPEAT or XFOR unless the corresponding loop is wholly included within the range of the PROC-CPROC construct.
- A procedure section may only be defined in a program after all PERFORMs to that PROC have occurred, i.e. PROCs normally preceed the END statement.
- A PERFORM in a FOR-loop causes non-standard FORTRAN to be generated.

2.7.3 Actions:

Whenever a PERFORM statement is encountered, the appropriate statements between PROC and CPROC are executed.

2.7.4 Example:

```
       .
       .
     PERFORM CLFILE
       .
       .
       .
       .
     PROC CLFILE
       FOR LUN=1,NUNITS
         CLOSE(UNIT=LUN)
       CFOR
     CPROC
```

2.8 I/O statements.

2.8.1 Form:

```
...,ERR=<exit specifier>,...
and/or
...,END=<exit specifier>,...
```

2.8.2 Notes:

- The following I/O statements are recognized: READ, WRITE, OPEN, CLOSE, REWIND, BACKSPACE, INQUIRE, ENDFILE.
- <exit specifier> can be any one of the following: XWHILE, XREPEAT, XFOR, STOP, RETURN or <procedure name>.
- The exit specification must appear on the first line of the I/O statement.

2.8.3 Actions:

The use of <procedure name> is equivalent to a PERFORM of the procedure between the I/O statement and the statement following it. The use of the other items will have the normal SHELTRAN meaning.

2.8.4 Example:

```
WHILE .TRUE.
  READ(1,FMT='(A)',END=XWHILE,ERR=ERRPRC) LINE
  WRITE(2,FMT='(1X,A)',ERR=ERRPRC) LINE
CWHILE
  .
  .
  .
PROC ERRPRC
  REWIND 1
  REWIND 2
CPROC
```

2.9 CALL statement.

Within the CALL argument list statement label arguments may be replaced by one of the same keywords for I/O statements. Such arguments must be preceded by an asterisk (*). E.g.

```
CALL sub ( ..., *XWHILE, ... )
```

Notes:

- This feature can only be used in calls to existing FORTRAN subroutines that specify alternate returns. The designers of SHELTRAN consider this feature undesirable and only provided an interface to existing routines, but have intentionally omitted the facility to construct them, i.e. the 'RETURN i' statement is not allowed.
- Asterisk arguments must appear on the first line of the CALL statement.

2.10 Additional features.

2.10.1 Comment lines.

Comment lines follow the same rules as FORTRAN.

2.10.2 Note lines.

Note lines are identified by an N in column 1, the text area being columns 2 to 72. In the line printer listing the text of a Note line is printed in columns 75 to 124 and on the same line as the statement following the Note line. If the next line is also a Note, the text is printed in the same columns of an otherwise blank line. The text of the Note line may be 50 characters long and may appear anywhere in columns 2 to 72. Leading blanks between column 1 and the start of the Note text are ignored, i.e. in the listing Notes are aligned.

2.10.3 Eject lines.

Eject lines are identified by an E in column 1, the text area being columns 7 to 72. The action of an Eject line is to force a new page on the line printer listing, with the page header taken from the Eject line text.

2.10.4 Fortran directive lines.

Fortran directive lines are identified by an F in column 1. The remainder of the line is written in columns 1 to 79 of the FORTRAN output file. This facility can be used to pass directives to the FORTRAN compiler (e.g. debug directives). It should not be used to bypass SHELTRAN control structures.

2.10.5 Input source deck.

The input SHELTRAN source deck follows the normal FORTRAN layout rules, apart from the above-mentioned features. No indentation of structures need to be included, as the pre-processor will automatically provide indentation in the listing.

2.10.6 Output source deck.

The output source deck consists of the input source deck plus the necessary generated FORTRAN statements. It contains no Comment, Eject or Note lines. The output deck is not indented.

2.10.7 Include Files.

Code from other SHELTRAN sources can be included by putting an I in column 1 followed by the name of the source to be included. The code included in this way will also be processed by the SHELTRAN compiler.

2.11 Limitations.

- The maximum number of procedure sections in a program unit is 99.
- The following variable names are reserved by SHELTRAN and must not be used: IF, ELSEIF, WHILE, UNTIL, FOR; all 6-character variable names beginning with 'IZZZ'
- It is not allowed to give variable names beginning with 'I' an implicit type other than INTEGER.
- PERFORM statements in FOR-loops cause non-standard FORTRAN to be generated. This is because after a procedure section has been executed, a jump into the generated DO-loop is necessary to resume execution after the PERFORM statement. Most compilers however allow this kind of jump which was part of the previous FORTRAN standard (FORTRAN IV).

Chapter 3

Update history.

```
 Jan-83  version 5.0: completely revised for FORTRAN 77
                 J.P. Terlouw

21-Mar-83: scratch file opening changed; instead of
    TAPE2 and TAPE3 we now use SCRAT01 and SCRAT02
               J.P.T.

 3-Apr-84: error message for non-standard Fortran when using
    PERFORM in for loops suppressed.
               J.P.T.

 1-Nov-88: bug in FOR loop control repaired, procedure names
    upto 31 characters, no trailing blanks in target and
    source listing.
               K.G.B.
```