GENTRAN User's Manual
REDUCE Version

Barbara L. Gates
RAND
Santa Monica CA 90407-2138 USA
Updated for REDUCE 3.4 by
Michael C. Dewar
The University of Bath
Email: miked@nag.co.uk

February 1991

GENTRAN is an automatic code GENerator and TRANslator which runs under REDUCE and VAXIMA. It constructs complete numerical programs based on sets of algorithmic specifications and symbolic expressions. Formatted FORTRAN, RATFOR or C code can be generated through a series of interactive commands or under the control of a template processing routine. Large expressions can be automatically segmented into subexpressions of manageable size, and a special file-handling mechanism maintains stacks of open I/O channels to allow output to be sent to any number of files simultaneously and to facilitate recursive invocation of the whole code generation process. GENTRAN provides the flexibility necessary to handle most code generation applications. This manual describes usage of the GENTRAN package for REDUCE.

Acknowledgements

The GENTRAN package was created at Kent State University to generate numerical code for computations in finite element analysis. I would like to thank Prof. Paul Wang for his guidance and many suggestions used in designing the original package for VAXIMA.

The second version of GENTRAN was implemented at Twente University of Technology to run under REDUCE. It was designed to be interfaced with a code optimization facility created by Dr. J. A. van Hulzen. I would like to thank Dr. van Hulzen for all of his help in the implementation of GENTRAN in RLISP during a stay at his university in The Netherlands.

Finally, I would like to thank Dr. Anthony Hearn of the RAND Corporation for his help in better integrating GENTRAN into the REDUCE environment.

1 INTRODUCTION

Solving a problem in science or engineering is often a two-step process. First the problem is modeled mathematically and derived symbolically to provide a set of formulas which describe how to solve the problem numerically. Next numerical programs are written based on this set of formulas to efficiently compute specific values for given sets of input. Computer algebra systems such as REDUCE provide powerful tools for use in the formula-derivation phase but only provide primitive program-coding tools. The GENTRAN package [**?**, **?**, **?**, **?**] has been constructed to automate the tedious, time consuming and error-prone task of writing numerical programs based on a set of symbolic expressions.

1.1 The GENTRAN Code Generator and Translator

The GENTRAN code GENeration and TRANslation package, originally implemented in Franz LISP to run under VAXIMA [**?**], is now also implemented in RLISP to run under REDUCE. Although GENTRAN was originally created specifically to generate numerical code for use with an existing FORTRAN-based finite element analysis package [**?**, **?**], it was designed to provide the flexibility required to handle most code generation applications. GENTRAN contains code generation commands, file-handling commands, mode switches, and global variables, all of which are accessible from both the algebraic and symbolic modes of REDUCE to give the user maximal control over the code generation process. Formatted FORTRAN [**?**], RATFOR [**?**], C [**?**], or PASCAL code can be generated from algorithmic specifications, i.e., descriptions of the behaviour of the target numerical program expressed

in the REDUCE programming language, and from symbolically derived expressions and formulas.

In addition to arithmetic expressions and assignment statements, GENTRAN can also generate type declarations and control-flow structures. Code generation can be guided by user-supplied template file(s) to insert generated code into pre-existing program skeletons, or it can be accomplished interactively through a series of translation commands without the use of template files. Special mode switches enable the user to turn on or off specific features such as automatic segmentation of large expressions, and global variables allow the user to modify the code formatting process. Generated code can be sent to one or more files and, optionally, to the user's terminal. Stacks of open I/O channels facilitate temporary output redirection and recursive invocation of the code generation process.

1.2 Code Optimization

A code optimizer [**?**], which runs under REDUCE, has been constructed to reduce the arithmetic complexity of a set of symbolic expressions (see the SCOPE package on page **??**). It optimizes them by extracting common subexpressions and assigning them to temporary variables which are inserted in their places. The optimization technique is based on mapping the expressions onto a matrix of coefficients and exponents which are searched for patterns corresponding to the common subexpressions. Due to this process the size of the expressions is often considerably reduced.

GENTRAN and the Code Optimizer have been interfaced to make it possible to generate optimized numerical programs directly from REDUCE. Setting the switch `GENTRANOPT` **ON** specifies that all sequences of assignment statements are to be optimized before being converted to numerical code.

1.3 Organization of the Manual

The remainder of this manual is divided into five sections. Sections 2 and 3 describe code generation. Section 2 explains interactive code generation, the expression segmentation facility, and how temporary variables can be generated; then section 3 explains how code generation can be guided by a user-supplied template file. Section 4 describes methods of output redi-

rection, and section 5 describes user-accessible global variables and mode switches which alter the code generation process. Finally section 6 presents three complete examples.

1.3.1 Typographic Conventions

The following conventions are used in the syntactic definitions of commands in this manual:

- Command parts which must be typed exactly as shown are given in **BOLD PRINT**.
- User-supplied arguments are *emphasized*.
- [...] indicate optional command parts.

The syntax of each GENTRAN command is shown terminated with a **;**. However, either **;** or **$** can be used to terminate any command with the usual REDUCE meaning: **;** indicates that the returned value is to be printed, while **$** indicates that printing of the returned value is to be suppressed.

Throughout this manual it is stated that file name arguments must be atoms. The exact type of atom (e.g., identifier or string) is system and/or site dependent. The instructions for the implementation being used should therefore be consulted.

2 Interactive Code Generation

GENTRAN generates numerical programs based on algorithmic specifications in the REDUCE programming language and derived symbolic expressions produced by REDUCE evaluations. FORTRAN, RATFOR, PASCAL or C code can be produced. Type declarations can be generated, and comments and other literal strings can be inserted into the generated code. In addition, large arithmetic expressions can be automatically segmented into a sequence of subexpressions of manageable size.

This section explains how to select the target language, generate code, control expression segmentation, and how to generate temporary variable names.

2.1 Target Language Selection

Before generating code, the target numerical language must be selected. GENTRAN is currently able to generate FORTRAN, RATFOR, PASCAL and C code. The global variable **GENTRANLANG!*** determines which type of code is produced. **GENTRANLANG!*** can be set in algebraic or symbolic mode. It can be set to any value, but only four atoms have special meaning: **FORTRAN**, **RATFOR**, **PASCAL** and **C**. Any other value is assumed to mean **FORTRAN**. **GENTRANLANG!*** is always initialized to **FORTRAN**.

2.2 Translation

The **GENTRAN** (GENerate/TRANslate) command is used to generate numerical code and also to translate code from algorithmic specifications in the REDUCE programming language to code in the target numerical language. Section 2.4 explains code *generation*. This section explains code *translation*.

A substantial subset of all expressions and statements in the REDUCE programming language can be translated directly into numerical code. The **GENTRAN** command takes a REDUCE expression, statement, or procedure definition, and translates it into code in the target language.

Syntax:

GENTRAN *stmt* [**OUT** *f1,f2,... ,fn*];

Arguments:

stmt is any REDUCE expression, statement (simple, compound, or group), or procedure definition that can be translated by GENTRAN into the target language[1] *stmt* may contain any number of calls to the special functions **EVAL**, **DECLARE**, and **LITERAL** (see sections 2.2 – 2.8).

f1,f2,... ,fn is an optional argument list containing one or more *f*'s, where each *f* is one of:

[1] See 8 for a complete listing of REDUCE expressions and statements that can be translated.

an atom	=	an output file
T	=	the terminal
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN (see section 4)

Side Effects:

GENTRAN translates *stmt* into formatted code in the target language.

If the optional part of the command is not given, generated code is simply written to the current output file. However, if it is given, then the current output file is temporarily overridden. Generated code is written to each file represented by *f1,f2,...,fn* for this command only. Files which were open prior to the call to **GENTRAN** will remain open after the call, and files which did not exist prior to the call will be created, opened, written to, and closed. The output stack will be exactly the same both before and after the call.

Returned Value:

GENTRAN returns the name(s) of the file(s) to which code was written.

Diagnostic Messages:

```
*** OUTPUT FILE ALREADY EXISTS
    OVERWRITE FILE? (Y/N)
```

```
***** WRONG TYPE OF ARG
```

exp

```
***** CANNOT BE TRANSLATED
```

Example 1[2]

```
1: GENTRANLANG!* := 'FORTRAN$

2: GENTRAN
2:     FOR I:=1:N DO
```

[2]When the **PERIOD** flag (default setting: ON) is turned on, all integers are automatically printed as real numbers except exponents, subscripts in subscripted variables, and index values in DO-type loops.

```
2:         V(I) := 0$

      DO 25001 I=1,N
         V(I)=0.0
25001 CONTINUE

3: GENTRANLANG!* := 'RATFOR$

4: GENTRAN
4:      FOR I:=1:N DO
4:          FOR J:=I+1:N DO
4:          <<
4:              X(J,I) := X(I,J);
4:              Y(J,I) := Y(I,J)
4:          >>$

DO I=1,N
    DO J=I+1,N
    {
        X(J,I)=X(I,J)
        Y(J,I)=Y(I,J)
    }

5: GENTRANLANG!* := 'C$

6: GENTRAN
6:      P := FOR I:=1:N PRODUCT I$

{
    P=1;
    for (I=1;I<=N;++I)
        P*=I;
}

7: GENTRANLANG!* := 'PASCAL$

8: GENTRAN
8:     S := FOR I := 1:10 SUM V(I)$
BEGIN
```

```
        S:=0;
        FOR I:=1 TO 10 DO
            S:=S+V(I)
    END;
```

Translation is a convenient method of producing numerical code when the exact behaviour of the resultant code is known. It gives the REDUCE user who is familiar with the syntax of statements in the REDUCE programming language the ability to write code in a numerical programming language without knowing the exact syntactical requirements of the language. However the *real* power of the **GENTRAN** command lies in its ability to generate code: it can produce numerical code from symbolic expressions derived in REDUCE in addition to translating statements directly. This aspect is described in section 2.4.

2.3 Precision

By default **GENTRAN** generates constants and type declarations in single precision form. If the user requires double precision output then the switch **DOUBLE** must be set **ON**. This does the following:

- Declarations of appropriate type are converted to their double precision counterparts. In FORTRAN and RATFOR this means that objects of type *REAL* are converted to objects of type *DOUBLE PRECISION* and objects of type *COMPLEX* are converted to *COMPLEX*16* [3]. In C the counterpart of *float* is *double*, and of *int* is *long*. There is no complex data type and trying to translate complex objects causes an error.
- Similarly subprograms are given their correct type where appropriate.
- In FORTRAN and RATFOR *REAL* and *COMPLEX* numbers are printed with the correct double precision format.
- Intrinsic functions are converted to their double precision counterparts (e.g. in FORTRAN $SIN \rightarrow DSIN$ etc.).

[3] This is not part of the ANSI FORTRAN standard. Some compilers accept *DOUBLE COMPLEX* as well as, or instead of, *COMPLEX*16*, and some accept neither.

2.3.1 Intrinsic FORTRAN and RATFOR functions.

An attempt is made to convert the arguments of intrinsic functions to the correct type. For example:

```
5: GENTRAN f:=sin(1)$
      F=SIN(1.0)

6: GENTRAN f:=sin(x)$
      F=SIN(REAL(X))

7: GENTRAN DECLARE <<x:real>>$

8: GENTRAN f:=sin(x)$
      F=SIN(X)
```

Which function is used to coerce the argument may, of course, depend on the setting of the switch **DOUBLE**.

2.3.2 Number of printed floating point digits.

To ensure the correct number of floating point digits are generated it may be necessary to use either the **PRECISION** or **PRINT!-PRECISION** commands. The former alters the number of digits REDUCE calculates, the latter only the number of digits REDUCE prints. Each takes an integer argument. It is not possible to set the printed precision higher than the actual precision. Calling **PRINT!-PRECISION** with a negative argument causes the printed precision to revert to the actual precision.

```
1: on rounded$

2: precision 16$

3: 1/3;

0.333 33333 33333 333

4: print!-precision 6$

5: 1/3;
```

```
0.333333

6: print!-precision(-1)$

7: 1/3;

0.333 33333 33333 333
```

2.4 Code Generation: Evaluation Prior to Translation

Section 2.2 showed how REDUCE statements and expressions can be translated directly into the target language. This section shows how to indicate that parts of those statements and expressions are to be handed to REDUCE to be evaluated before being translated. In other words, this section explains how to generate numerical code from algorithmic specifications (in the REDUCE programming language) and symbolic expressions. Each of the following four subsections describes a special function or operator that can be used to request partial or full evaluation of expressions prior to translation. Note that these functions and operators have the described effects *only* when applied to arguments to the **GENTRAN** function and that evaluation is done in algebraic or symbolic mode, depending on the value of the REDUCE variable **!*MODE**.

2.4.1 The EVAL Function

Syntax:

EVAL *exp*

Argument:

exp is any REDUCE expression or statement which, after evaluation by REDUCE, results in an expression that can be translated by GENTRAN into the target language.

Side Effect:

When **EVAL** is called on an expression which is to be translated, it tells **GENTRAN** to give the expression to REDUCE

for evaluation first, and then to translate the result of that evaluation.

Example 2

The following formula, F, has been derived symbolically:

```
   2
2*X  - 5*X + 6
```

We wish to generate an assignment statement for the quotient of F and its derivative.

```
1: GENTRAN
1:      Q := EVAL(F)/EVAL(DF(F,X))$

      Q=(2.0*X**2-(5.0*X)+6.0)/(4.0*X-5.0)
```

2.4.2 The :=: Operator

In many applications, assignments must be generated in which the left-hand side is some known variable name, but the right-hand side is an expression that must be evaluated. For this reason, a special operator is provided to indicate that the expression on the right-hand side is to be evaluated prior to translation. This special operator is **:=:** (i.e., the usual REDUCE assignment operator with an extra ":" on the right).

Example 3

```
1: GENTRAN
1:  DERIV :=: DF(X^4-X^3+2*x^2+1,X)$

       DERIV=4.0*X**3-(3.0*X**2)+4.0*X
```

Each built-in operator in REDUCE has an alternative alphanumeric identifier associated with it. Similarly, the GENTRAN **:=:** operator has a special identifier associated with it: **RSETQ** may be used interchangeably with **:=:** on input.

2.4.3 The ::= Operator

When assignments to matrix or array elements must be generated, many times the indices of the element must be evaluated first. The special operator

::= can be used within a call to **GENTRAN** to indicate that the indices of the matrix or array element on the left-hand side of the assignment are to be evaluated prior to translation. (This is the usual REDUCE assignment operator with an extra ":" on the left.)

Example 4

We wish to generate assignments which assign zeros to all elements on the main diagonal of M, an n x n matrix.

```
10: FOR j := 1 : 8 DO
10:      GENTRAN
10:          M(j,j) ::= 0$

      M(1,1)=0.0
      M(2,2)=0.0
      :
      :
      M(8,8)=0.0
```

LSETQ may be used interchangeably with **::=** on input.

2.4.4 The ::=: Operator

In applications in which evaluated expressions are to be assigned to array elements with evaluated subscripts, the **::=:** operator can be used. It is a combination of the **::=** and **:=:** operators described in sections 2.4.2 and 2.4.3.

Example 5

The following matrix, M, has been derived symbolically:

```
(  A   0  -1   1)
(                )
(  0   B   0   0)
(                )
( -1   0   C  -1)
(                )
(  1   0  -1   D)
```

We wish to generate assignment statements for those elements on the main diagonal of the matrix.

```
10: FOR j := 1 : 4 DO
10:      GENTRAN
10:          M(j,j) ::=: M(j,j)$

      M(1,1)=A
      M(2,2)=B
      M(3,3)=C
      M(4,4)=D
```

The alternative alphanumeric identifier associated with **::=:** is **LRSETQ**.

2.5 Explicit Type Declarations

Type declarations are automatically generated each time a subprogram heading is generated. Type declarations are constructed from information stored in the GENTRAN symbol table. The user can place entries into the symbol table explicitly through calls to the special GENTRAN function **DECLARE**.

Syntax:

DECLARE *v1,v2,. . . ,vn* **:** *type;*

or

DECLARE
<<
v11,v12,... ,v1n **:** *type1;*
v21,v22,... ,v2n **:** *type2;*
:
:
vn1,vnn,... ,vnn **:** *typen;*
>>;

Arguments:

Each *v1,v2,... ,vn* is a list of one or more variables (optionally subscripted to indicate array dimensions), or variable ranges (two letters separated by a "-"). *v*'s are not evaluated unless given as arguments to **EVAL**.

Each *type* is a variable type in the target language. Each must be an atom, optionally preceded by the atom **IMPLICIT**. *type*'s are not evaluated unless given as arguments to **EVAL**.

Side Effect:

Entries are placed in the symbol table for each variable or variable range declared in the call to this function. The function call itself is removed from the statement group being translated. Then after translation, type declarations are generated from these symbol table entries before the resulting executable statements are printed.

Diagnostic Message:

```
***** INVALID SYNTAX
```

Example 6

```
1: GENTRAN
1: <<
1:      DECLARE
1:      <<
1:          A-H, O-Z : IMPLICIT REAL;
1:          M(4,4)   : INTEGER
1:      >>;
1:      FOR I:=1:4 DO
1:          FOR J:=1:4 DO
```

```
1:              IF I=J
1:                  THEN M(I,J):=1
1:                  ELSE M(I,J):=0;
1:      DECLARE I, J : INTEGER;
1: >>$

      IMPLICIT REAL (A-H,O-Z)
      INTEGER M(4,4),I,J
      DO 25001 I=1,4
          DO 25002 J=1,4
              IF (I.EQ.J) THEN
                  M(I,J)=1.0
              ELSE
                  M(I,J)=0.0
              ENDIF
25002     CONTINUE
25001 CONTINUE
```

The **DECLARE** statement can also be used to declare subprogram types (i.e. **SUBROUTINE** or **FUNCTION**) for FORTRAN and RATFOR code, and function types for all four languages.

Example 7

```
1: GENTRANLANG!* := 'RATFOR$

2: GENTRAN
2:      PROCEDURE FAC N;
2:      BEGIN
2:      DECLARE
2:      <<
2:          FAC : FUNCTION;
2:          FAC, N : INTEGER
2:      >>;
2:      F := FOR I:=1:N PRODUCT I;
2:      DECLARE F, I : INTEGER;
2:      RETURN F
2:      END$

INTEGER FUNCTION FAC(N)
```

```
INTEGER N,F,I
{
    F=1
    DO I=1,N
        F=F*I
}
RETURN(F)
END

3: GENTRANLANG!* := 'C$
4: GENTRAN
4:      PROCEDURE FAC N;
4:      BEGIN
4:      DECLARE FAC, N, I, F : INTEGER;
4:      F := FOR I:=1:N PRODUCT I;
4:      RETURN F
4:      END$

int FAC(N)
int N;
{
    int I,F;
    {
        F=1;
        for (I=1;I<=N;++I)
            F*=I;
    }
    return(F);
}
```

When generating code for subscripted variables (i.e., matrix and array elements), it is important to keep several things in mind. First of all, when a REDUCE array is declared with a declaration such as

ARRAY A(*n*)$

where *n* is a positive integer, **A** is actually being declared to be of size **n**+1. Each of the elements **A(0), A(1), ... , A(n)** can be used. However, a FORTRAN or RATFOR declaration such as

DIMENSION A(*n*)

declares **A** only to be of size **n**. Only the elements **A(1), A(2), ... , A(n)** can be used. Furthermore, a C declaration such as

float A[*n*]**;**

declares **A** to be of size **n** with elements referred to as **A[0], A[1], ... , A**[*n-1*].

To resolve these array size and subscripting conflicts, the user should remember the following:

- *All REDUCE array subscripts are translated literally.* Therefore it is the user's responsibility to be sure that array elements with subscript 0 are not translated into FORTRAN or RATFOR.
- Since C and PASCAL arrays allow elements with a subscript of 0, when an array is declared to be of size *n* by the user, *the actual generated type declaration will be of size n+1* so that the user can translate elements with subscripts from 0, and up to and including *n*.

If the user is generating C code, it is possible to produce declarations for arrays with unknown bounds:

```
5: gentran declare <<x(*,*):real;y(*):integer>>$

6: gendecs nil;
float X[ ][ ];
int Y[ ];
```

2.6 Implicit Type Declarations

Some type declarations can be made automatically if the switch **GETDECS** is **ON**. In this case:

1. The indices of loops are automatically declared to be integers.
2. There is a global variable **DEFTYPE!***, which is the default type given to objects. Subprograms, their parameters, and local scalar objects are automatically assigned this type. Note that types such as REAL*8 or DOUBLE PRECISION should not be used as, if **DOUBLE** is on, then a default type of REAL will in fact be DOUBLE PRECISION anyway.
3. If GENTRAN is used to translate a REDUCE procedure, then it as-

signs objects declared **SCALAR** the type given by **DEFTYPE!***. Note that it is legal to use the commands **INTEGER** and **REAL** in the place of **SCALAR**, which allows the user to specify an actual type. The procedure may also be given a return type, in which case that is used as the default. For example:

```
2: on getdecs,gendecs$

3: GENTRAN
3: real procedure f(x);
3: begin integer n;real y;
3:     n := 4;
3:     y := n/(1+x)^2;
3:     return y;
3: end;
      REAL FUNCTION F(X)
      INTEGER N
      REAL X,Y
      N=4
      Y=N/(1.0+X)**2
      F=Y
      RETURN
      END
```

2.7 More about Type Declarations

A check is made on output to ensure that all types generated are legal ones. This is necessary since **DEFTYPE!*** can be set to anything. Note that **DEFTYPE!*** ought normally to be given a simple type as its value, such as REAL, INTEGER, or COMPLEX, since this will always be translated into the corresponding type in the target language on output.

An entry is removed from the symbol table once a declaration has been generated for it. The **KEEPDECS** switch (by default **OFF**) disables this, allowing a user to check the types of objects which GENTRAN has generated (useful if they are being generated automatically)

2.8 Comments and Literal Strings

Comments and other strings of characters can be inserted directly into the stream of generated code through a call to the special function **LITERAL**.

Syntax:

LITERAL *arg1,arg2,... ,argn;*

Arguments:

arg1,arg2,... ,argn is an argument list containing one or more *arg*'s, where each *arg* either is, or evaluates to, an atom. The atoms **TAB!*** and **CR!*** have special meanings. *arg*'s are not evaluated unless given as arguments to **EVAL**.

Side Effect:

This statement is replaced by the character sequence resulting from concatenation of the given atoms. Double quotes are stripped from all string type *arg*'s, and the reserved atoms **TAB!*** and **CR!*** are replaced by a tab to the current level of indentation, and an end-of-line character, respectively.

Example 8

Suppose N has value 10.

```
1: GENTRANLANG!* := 'FORTRAN$

2: GENTRAN
2: <<
2:      LITERAL
2:        "C",TAB!*,"--THIS IS A FORTRAN COMMENT--",CR!*,
2:        "C",CR!*;
2:      LITERAL
2:        TAB!*,"DATA N/",EVAL(N),"/",CR!*
2: >>$

C     --THIS IS A FORTRAN COMMENT--
C
      DATA N/10/

3: GENTRANLANG!* := 'RATFOR$
```

```
4: GENTRAN
4:     FOR I:=1:N DO
4:     <<
4:         LITERAL
4:            TAB!*,"# THIS IS A RATFOR COMMENT",CR!*;
4:         LITERAL
4:            TAB!*,"WRITE(6,10) (M(I,J),J=1,N)",CR!*,
4:            10,TAB!*,"FORMAT(1X,10(I5,3X))",CR!*
4:     >>$

DO I=1,N
    {
        # THIS IS A RATFOR COMMENT
        WRITE(6,10) (M(I,J),J=1,N)
10      FORMAT(1X,10(I5,3X))
    }

5: GENTRANLANG!* := 'C$
6: GENTRAN
6: <<
6:      X:=0;
6:      LITERAL "/* THIS IS A",CR!*,
6:              "   C COMMENT */",CR!*
6: >>$

{
    X=0.0;
/* THIS IS A
   C COMMENT */
}

7:  GENTRANLANG!* := 'PASCAL$

8: GENTRAN
8: <<
8:     X := SIN(Y);
8:     LITERAL "{ THIS IS A PASCAL COMMENT }", CR!*
8: >>$
BEGIN
```

```
    X:=SIN(Y)
{ THIS IS A PASCAL COMMENT }
END;
```

2.9 Expression Segmentation

Symbolic derivations can easily produce formulas that can be anywhere from a few lines to several pages in length. Such formulas can be translated into numerical assignment statements, but unless they are broken into smaller pieces they may be too long for a compiler to handle. (The maximum number of continuation lines for one statement allowed by most FORTRAN compilers is only 19.) Therefore GENTRAN contains a segmentation facility which automatically *segments*, or breaks down unreasonably large expressions.

The segmentation facility generates a sequence of assignment statements, each of which assigns a subexpression to an automatically generated temporary variable. This sequence is generated in such a way that temporary variables are re-used as soon as possible, thereby keeping the number of automatically generated variables to a minimum. The facility can be turned on or off by setting the mode switch **GENTRANSEG** accordingly (i.e., by calling the REDUCE function **ON** or **OFF** on it). The user can control the maximum allowable expression size by setting the variable **MAXEXPPRINTLEN!*** to the maximum number of characters allowed in an expression printed in the target language (excluding spaces automatically printed by the formatter). The **GENTRANSEG** switch is on initially, and **MAXEXPPRINTLEN!*** is initialized to 800.

Example 9

```
1: ON EXP$

2: JUNK1 := (A+B+C+D)^2$

3: MAXEXPPRINTLEN!* := 24$

4: GENTRAN VAL :=: JUNK1$

      T0=A**2+2.0*A*B
```

```
        T0=T0+2.0*A*C+2.0*A*D
        T0=T0+B**2+2.0*B*C
        T0=T0+2.0*B*D+C**2
        VAL=T0+2.0*C*D+D**2

5: JUNK2 := JUNK1/(E+F+G)$

6: MAXEXPPRINTLEN!* := 23$

7: GENTRANLANG!* := 'C$

8: GENTRAN VAL :=: JUNK2$

{
    T0=power(A,2)+2.0*A*B;
    T0+=2.0*A*C;
    T0=T0+2.0*A*D+power(B,2);
    T0+=2.0*B*C;
    T0=T0+2.0*B*D+power(C,2);
    T0=T0+2.0*C*D+power(D,2);
    VAL=T0/(exp(1.0)+F+G);
}
```

2.9.1 Implicit Type Declarations

When the segmentation routine generates temporary variables, it places type declarations in the symbol table for those variables if possible. It uses the following rules to determine their type:

(1) If the type of the variable to which the large expression is being assigned is already known (i.e., has been declared by the user), then the temporary variables will be declared to be of that same type.

(2) If the global variable **TEMPVARTYPE!*** has a non-NIL value, then the temporary variables are declared to be of that type.

(3) Otherwise, the variables are not declared.

Example 10

```
1: MAXEXPPRINTLEN!* := 20$

2: TEMPVARTYPE!* := 'REAL$

3: GENTRAN
3: <<
3:      DECLARE ISUM : INTEGER;
3:      ISUM := II+JJ+2*KK+LL+10*MM+NN;
3:      PROD := V(X,Y)*SIN(X)*COS(Y^2)*(X+Y+Z^2)
3: >>$

      INTEGER ISUM,T0
      REAL T1
      T0=II+JJ+2.0*KK+LL
      ISUM=T0+10.0*MM+NN
      T1=V(X,Y)*SIN(X)*COS(Y**2)
      PROD=T1*(X+Y+Z**2)
```

2.10 Generation of Temporary Variable Names

As we have just seen, GENTRAN's segmentation module generates temporary variables and places type declarations in the symbol table for them whenever possible. Various other modules also generate variables and corresponding declarations. All of these modules call one special GENTRAN function each time they need a temporary variable name. This function is **TEMPVAR**. There are situations in which it may be convenient for the user to be able to generate temporary variable names directly.[4] Therefore **TEMPVAR** is a user-accessible function which may be called from both the algebraic and symbolic modes of REDUCE.

Syntax:

TEMPVAR *type*

Argument:

[4]One such example is suppression of the simplification process to generate numerical code which is more efficient. See the example in section 6.2.3 on page 68.

type is an atom which either indicates the variable type in the target language (INTEGER, REAL, etc.), or is **NIL** if the variable type is unknown.

Side Effects:

TEMPVAR creates temporary variable names by repeatedly concatenating the values of the global variables **TEMPVARNAME!*** (which has a default value of **T**) and **TEMPVARNUM!*** (which is initially set to 0) and incrementing **TEMPVARNUM!*** until a variable name is created which satisfies one of the following conditions:

(1) It was not generated previously, and it has not been declared by the user.

(2) It was previously generated to hold the same type of value that it must hold this time (e.g. INTEGER, REAL, etc.), and the value assigned to it previously is no longer needed.

If *type* is a non-NIL argument, or if *type* is **NIL** and the global variable **TEMPVARTYPE!*** (initially NIL) has been set to a non-NIL value, then a type entry for the generated variable name is placed in the symbol table.

Returned Value:

TEMPVAR returns an atom which can be used as a variable.

Note: It is the user's responsibility to set **TEMPVARNAME!*** and **TEMPVARNUM!*** to values such that generated variable names will not clash with variables used elsewhere in the program unless those variables have been declared.

2.10.1 Marking Temporary Variables

In section 2.10 we saw that a temporary variable name (of a certain type) can be regenerated when the value previously assigned to it is no longer needed. This section describes a function which *marks* a variable to indicate that it currently holds a significant value, and the next section describes functions which *unmark* variables to indicate that the values they hold are no longer significant.

Syntax:

MARKVAR *var*

Argument:

var is an atom.

Side Effects:

MARKVAR sets a flag on *var*'s property list to indicate that *var* currently holds a significant value.

Returned Value:

MARKVAR returns *var*.

Example 11

The following matrix, M has been derived symbolically:

```
(X*(Y+Z)       0     X*Z)
(                      )
(     -X     X+Y       0)
(                      )
(    X*Z       0    Z**2)
```

We wish to replace each non-zero element by a generated variable name to prevent these expressions from being resubstituted into further calculations. (We will also record these substitutions in the numerical program we are constructing by generating assignment statements.)[5]

```
9: SHARE var$

10: FOR j := 1 : 3 DO
10:       FOR k := 1 : 3 DO
10:           IF M(j,k) NEQ 0 THEN
10:           <<
10:               var := TEMPVAR(NIL);
10:               MARKVAR var;
10:               GENTRAN
10:                   EVAL(var) :=: M(j,k);
```

[5]Note: **MARKVAR** is a symbolic mode procedure. Therefore, the name of each variable whose value is to be passed to it from algebraic mode must appear in a **SHARE** declaration. This tells REDUCE to share the variable's value between algebraic and symbolic modes.

```
10:              M(j,k) := var
10:          >>$

      T0=X*(Y+Z)
      T1=X*Z
      T2=-X
      T3=X+Y
      T4=X*Z
      T5=Z**2
```

Now matrix M contains the following entries:

```
(T0   0  T1)
(          )
(T2  T3   0)
(          )
(T4   0  T5)
```

2.10.2 Unmarking Temporary Variables

After the value assigned to a temporary variable has been used in the numerical program and is no longer needed, the variable name can be *unmarked* with the **UNMARKVAR** function.

Syntax:

UNMARKVAR *var;*

Argument:

var is an atom (variable name) or an expression containing one or more variable names.

Side Effect:

UNMARKVAR resets flags on the property lists of all variable names in *var* to indicate that they do not hold significant values any longer.

2.11 Enabling and Disabling Generation of Type Declarations

GENTRAN maintains a symbol table of variable type and dimension information. It adds information to the symbol table by processing user-supplied calls to the **DECLARE** function (see Section 2.5) and as a side effect of generating temporary variable names (see Sections 2.9 and 2.10). All information is stored in the symbol table until GENTRAN is ready to print formatted numerical code. Since programming languages such as FORTRAN require that type declarations appear before executable statements, GENTRAN automatically extracts all relevant type information and prints it in the form of type declarations before printing executable statements. This feature is useful when the entire body of a (sub)program is generated at one time: in this case, type declarations are printed before any executable code. However, if the user chooses to generate code in pieces, the resulting code may have type declarations interleaved with executable code. For this reason, the user may turn the **GENDECS** mode switch on or off, depending on whether or not s/he chooses to use this feature.

In the following we re-examine the example of Section 2.9.1.

Example 12

```
1: MAXEXPPRINTLEN!* := 20$

2: TEMPVARTYPE!* := 'REAL!*8$

3: GENTRAN
3: <<
3:      DECLARE ISUM : INTEGER;
3:      ISUM := II+JJ+2*KK+LL+10*MM+NN
3: >>$

      INTEGER ISUM,T0
      T0=II+JJ+2*KK+LL
      ISUM=T0+10*MM+NN

4: GENTRAN PROD := V(X,Y)*SIN(X)*COS(Y^2)*(X+Y+Z^2)$

      REAL*8 T2
```

```
      T2=V(X,Y)*SIN(REAL(X))*COS(REAL(Y**2))
      PROD=T2*(X+Y+Z**2)

5: OFF GENDECS$

6: GENTRAN
6: <<
6:      DECLARE ISUM : INTEGER;
6:      ISUM := II+JJ+2*KK+LL+10*MM+NN
6: >>$

      T0=II+JJ+2*KK+LL
      ISUM=T0+10*MM+NN

7: GENTRAN PROD := V(X,Y)*SIN(X)*COS(Y^2)*(X+Y+Z^2)$

      T2=V(X,Y)*SIN(REAL(X))*COS(REAL(Y**2))
      PROD=T2*(X+Y+Z**2)
```

In Section 3.4 we will explain how to further control the generation of type declarations.

2.12 Complex Numbers

With the switch **COMPLEX** set **ON**, GENTRAN will generate the correct representation for a complex number in the given precision provided that:

1. The current language supports a complex data type (if it doesn't then an error results);
2. The complex quantity is evaluated by REDUCE to give an object of the correct domain; i.e.

   ```
   GENTRAN x:=: 1+i;

   GENTRAN x:= eval 1+i;

   z := 1+i;
   GENTRAN x:=: z;
   ```

   will all generate the correct result, as will their Symbolic mode equiv-

alents, while:

```
GENTRAN x := 1+i;
```

will not.

2.13 Intrinsic Functions

A warning is issued if a standard REDUCE function is encountered which does not have an intrinsic counterpart in the target language (e.g. *cot*, *sec* etc.). Output is not halted in case this is a user–supplied function, either via a REDUCE definition or within a GENTRAN template.

The types of intrinsic FORTRAN functions are coerced to reals (in the correct precision) as the following examples demonstrate:

```
19: GENTRAN x:=sin(0)$
      X=SIN(0.0)

20: GENTRAN x:=cos(A)$
      X=COS(REAL(A))

21: ON DOUBLE$

22: GENTRAN x := log(1)$
      X=DLOG(1.0D0)

23: GENTRAN x := exp(B)$
      X=DEXP(DBLE(B))

24: GENTRAN DECLARE <<b:real>>$

25: GENTRAN x := exp(B)$
      X=DEXP(B)
```

2.14 Miscellaneous

2.14.1 MAKECALLS

A statement like:

```
GENTRAN x^2+1$
```

will yield the result:

```
X**2+1
```

but, under normal circumstances, a statement like:

```
GENTRAN sin(x)$
```

will yield the result:

```
CALL SIN(X)
```

The switch **MAKECALLS** (OFF by default) will make GENTRAN yield

```
SIN(X)
```

This is useful if you don't know in advance what the form of the expression which you are translating is going to be.

2.14.2 E

When GENTRAN encounters e it translates it into EXP(1), and when GENTRAN encounters e^x it is translated to EXP(X). This is then translated into the correct statement in the given language and precision. Note that it is still possible to do something like:

```
GENTRAN e:=:e;
```

and get the correct result.

2.15 Booleans

Some languages, like Fortran-77, have a boolean data type. Others, like C, do not. When translating Reduce code into a language with a boolean data type, GENTRAN will recognise the special identifiers *true* and $false$. For example:

```
3: gentran <<declare t:logical ;t:=true>>;
      LOGICAL T
      T=.TRUE.
```

3 Template Processing

In some code generation applications pieces of the target numerical program are known in advance. A *template* file containing a program outline is supplied by the user, and formulas are derived in REDUCE, converted to numerical code, and inserted in the corresponding places in the program outline to form a complete numerical program. A template processor is provided by GENTRAN for use in these applications.

3.1 The Template Processing Command

Syntax:

GENTRANIN *f1,f2,... ,fm* [**OUT** *f1,f2,... ,fn*]*;*

Arguments:

f1,f2,... ,fm is an argument list containing one or more *f*'s, where each *f* is one of:

an atom	=	a template (input) file
T	=	the terminal

f1,f2,... ,fn is an optional argument list containing one or more *f*'s, where each *f* is one of:

an atom	=	an output file
T	=	the terminal
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN (see section 4)

Side Effects:

GENTRANIN processes each template file *f1,f2,... ,fm* sequentially.

A template file may contain any number of parts, each of which is either an active or an inactive part. All active parts start

with the character sequence **;BEGIN;** and end with **;END;**. The end of the template file is indicated by an extra **;END;** character sequence.

Inactive parts of template files are assumed to contain code in the target language (FORTRAN, RATFOR, PASCAL or C, depending on the value of the global variable **GENTRANLANG!***). All inactive parts are copied to the output. Comments delimited by the appropriate characters,

C ... <cr>	FORTRAN (beginning in column 1)
# ... <cr>	RATFOR
/* ... */	C
{ ... **}** or ***(** ... **)***	PASCAL

are also copied in their entirety to the output. Thus the character sequences **;BEGIN;** and **;END;** have no special meanings within comments.

Active parts may contain any number of REDUCE expressions, statements, and commands. They are not copied directly to the output. Instead, they are given to REDUCE for evaluation in algebraic mode[6]. All output generated by each evaluation is sent to the output file(s). Returned values are only printed on the terminal.

Active parts will most likely contain calls to **GENTRAN** to generate code. This means that the result of processing a template file will be the original template file with all active parts replaced by generated code.

If **OUT** *f1,f2,... ,fn* is not given, generated code is simply written to the current-output file.

However, if **OUT** *f1,f2,... ,fn* is given, then the current-output file is temporarily overridden. Generated code is written to each file represented by *f1,f2,... ,fn* for this command only. Files which were open prior to the call to **GENTRANIN** will remain open after the call, and files which did not exist prior to the call

[6] Active parts are evaluated in algebraic mode unless the mode is explicitly changed to symbolic from within the active part itself. This is true no matter which mode the system was in when the template processor was called.

will be created, opened, written to, and closed. The output-stack will be exactly the same both before and after the call.

Returned Value:

GENTRANIN returns the names of all files written to by this command.

Diagnostic Messages:

```
*** OUTPUT FILE ALREADY EXISTS
    OVERWRITE FILE? (Y/N)

***** NONEXISTENT INPUT FILE

***** TEMPLATE FILE ALREADY OPEN FOR INPUT

***** WRONG TYPE OF ARG
```

Example 13

Suppose we wish to generate a FORTRAN subprogram to compute the determinant of a 3 x 3 matrix. We can construct a template file with an outline of the FORTRAN subprogram and REDUCE and GENTRAN commands to fill it in:

Contents of file `det.tem`:

```
      REAL FUNCTION DET(M)
      REAL M(3,3)
;BEGIN;
      OPERATOR M$
      MATRIX MM(3,3)$
      MM := MAT( (M(1,1),M(1,2),M(1,3)),
                 (M(2,1),M(2,2),M(2,3)),
                 (M(3,1),M(3,2),M(3,3)) )$
      GENTRAN DET :=: DET(MM)$
;END;
      RETURN
      END
;END;
```

Now we can generate a FORTRAN subprogram with the following REDUCE session:

```
1: GENTRANLANG!* := 'FORTRAN$

2: GENTRANIN
2:      "det.tem"
2: OUT "det.f"$
```

Contents of file det.f:

```
      REAL FUNCTION DET(M)
      REAL M(3,3)
      DET=M(3,3)*M(2,2)*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)
     . *M(2,3)*M(1,1))+M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1
     . ,2)-(M(3,1)*M(2,2)*M(1,3))
      RETURN
      END
```

3.2 Copying Files into Template Files

Template files can be copied into other template files with recursive calls to **GENTRANIN** ; i.e., by calling **GENTRANIN** from the active part of a template file.

For example, suppose we wish to copy the contents of a file containing a subprogram into a file containing a main program. We will call **GENTRANIN** to do the copying, so the subprogram file must have **;END;** on its last line:

Contents of file det.tem:

```
      REAL FUNCTION DET(M)
      REAL M(3,3)
      DET=M(3,3)*M(2,2)*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)
     . *M(2,3)*M(1,1))+M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1
     . ,2)-(M(3,1)*M(2,2)*M(1,3))
      RETURN
      END
;END;
```

Now the template file for the main program can be constructed with an active part which will include file det.tem:

Contents of file main.tem:

```
C
C  MAIN PROGRAM
```

```
C
      REAL M(3,3),DET
      WRITE(6,*) 'ENTER 3 x 3 MATRIX'
      DO 100 I=1,3
          READ(5,*) (M(I,J),J=1,3)
100   CONTINUE
      WRITE(6,*) ' DET = ', DET(M)
      STOP
      END
C
C  DETERMINANT CALCULATION
C
;BEGIN;
      GENTRANIN "det.tem"$
;END;
;END;
```

The following REDUCE session will create the file `main.f`:

```
1: GENTRANIN
1:      "main.tem"
1: OUT "main.f"$
```

Contents of file `main.f`:

```
C
C  MAIN PROGRAM
C
      REAL M(3,3),DET
      WRITE(6,*) 'ENTER 3 x 3 MATRIX'
      DO 100 I=1,3
          READ(5,*) (M(I,J),J=1,3)
100   CONTINUE
      WRITE(6,*) ' DET = ', DET(M)
      STOP
      END
C
C  DETERMINANT CALCULATION
C
      REAL FUNCTION DET(M)
      REAL M(3,3)
```

```
      DET=M(3,3)*M(2,2)*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)
     . *M(2,3)*M(1,1))+M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*
     . M(1,2)-(M(3,1)*M(2,2)*M(1,3))
      RETURN
      END
```

3.3 The Template File Stack

The REDUCE **IN** command takes one or more file names as arguments. REDUCE reads each of the given files and executes all statements and commands, any of which may be another **IN** command. A stack of input file names is maintained by REDUCE to allow recursive invocation of the **IN** command. Similarly, a stack of template file names is maintained by GENTRAN to facilitate recursive invocation of the template processor. Section 3.2 showed that the **GENTRANIN** command can be called recursively to copy files into other files. This section shows that template files which are copied into other template files can also contain active parts, and thus the whole code generation process can be invoked recursively.

We can generalize the example of section 3.2 by generating code recursively. We can extend it to generate code which will compute entries of the inverse matrix, also. Suppose we have created the file init.red, which contains REDUCE commands to create an nxn matrix MM and initialize its entries to M(1,1), M(1,2), ... , M(n, n), for some user-entered value of n:

Contents of file `init.red`:

```
OPERATOR M$
MATRIX MM(n,n)$
FOR J := 1 : n DO
    FOR K := 1 : n DO
        MM(J,K) := M(J,K)$
END$
```

We have also created template files `det.tem` and `inv.tem` which contain outlines of FORTRAN subprograms to compute the determinant and inverse of an nxn matrix, respectively:

Contents of file `det.tem`:

```
      REAL FUNCTION DET(M)
;BEGIN;
```

```
      GENTRAN
      <<
          DECLARE M(EVAL(n),EVAL(n)) : REAL;
          DET :=: DET(MM)
      >>$
;END;
      RETURN
      END
;END;
```

Contents of file `inv.tem`:

```
      SUBROUTINE INV(M,MINV)
;BEGIN;
      GENTRAN
      <<
          DECLARE M(EVAL(n),EVAL(n)),
          MINV(EVAL(n),EVAL(n)) : REAL;
          MINV :=: MM^(-1)
      >>$
;END;
      RETURN
      END
;END;
```

Now we can construct a template file with a generalized version of the main program given in section 3.2 and can place **GENTRANIN** commands in this file to generate code recursively from the template files det.tem and inv.tem:

Contents of file `main.tem`:

```
C
C  MAIN PROGRAM
C
;BEGIN;
      GENTRAN
      <<
          DECLARE
          <<
              M(EVAL(n),EVAL(n)),
              DET,
```

```
              MINV(EVAL(n),EVAL(n)) : REAL;
              N                     : INTEGER
          >>;
          LITERAL TAB!*, "DATA N/", EVAL(n), "/", CR!*
      >>$
;END;
      WRITE(6,*) 'ENTER ', N, 'x', N, ' MATRIX'
      DO 100 I=1,N
          READ(5,*) (M(I,J),J=1,N)
100   CONTINUE
      WRITE(6,*) ' DET = ', DET(M)
      WRITE(6,*) ' INVERSE MATRIX:'
      CALL INV(M,MINV)
      DO 200 I=1,N
          WRITE(6,*) (MINV(I,J),J=1,N)
200   CONTINUE
      STOP
      END
C
C  DETERMINANT CALCULATION
C
;BEGIN;
      GENTRANIN "det.tem"$
;END;
C
C  INVERSE CALCULATION
C
;BEGIN;
      GENTRANIN "inv.tem"$
;END;
;END;
```

The following REDUCE session will create the file `main.f`:

```
1: n := 3$

2: IN "init.red"$

3: GENTRANLANG!* := 'FORTRAN$
```

```
4: GENTRANIN
4:      "main.tem"
4: OUT "main.f"$
```

Contents of file `main.f`:

```
C
C  MAIN PROGRAM
C
      REAL M(3,3),DET,MINV(3,3)
      INTEGER N
      DATA N/3/
      WRITE(6,*) 'ENTER ', N, 'x', N, ' MATRIX'
      DO 100 I=1,N
          READ(5,*) (M(I,J),J=1,N)
100   CONTINUE
      WRITE(6,*) ' DET = ', DET(M)
      WRITE(6,*) ' INVERSE MATRIX:'
      CALL INV(M,MINV)
      DO 200 I=1,N
          WRITE(6,*) (MINV(I,J),J=1,N)
200   CONTINUE
      STOP
      END
C
C  DETERMINANT CALCULATION
C
      REAL FUNCTION DET(M)
      REAL M(3,3)
      DET=M(3,3)*M(2,2)*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)
     . *M(2,3)*M(1,1))+M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)
     . *M(1,2)-(M(3,1)*M(2,2)*M(1,3))
      RETURN
      END
C
C  INVERSE CALCULATION
C
      SUBROUTINE INV(M,MINV)
      REAL M(3,3),MINV(3,3)
```

```
      MINV(1,1)=(M(3,3)*M(2,2)-(M(3,2)*M(2,3)))/(M(3,3)*M(2,2
     . )*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1))
     . +M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(2
     . ,2)*M(1,3)))
      MINV(1,2)=(-(M(3,3)*M(1,2))+M(3,2)*M(1,3))/(M(3,3)*M(2,
     . 2)*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1)
     . )+M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(
     . 2,2)*M(1,3)))
      MINV(1,3)=(M(2,3)*M(1,2)-(M(2,2)*M(1,3)))/(M(3,3)*M(2,2
     . )*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1))
     . +M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(2
     . ,2)*M(1,3)))
      MINV(2,1)=(-(M(3,3)*M(2,1))+M(3,1)*M(2,3))/(M(3,3)*M(2,
     . 2)*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1)
     . )+M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(
     . 2,2)*M(1,3)))
      MINV(2,2)=(M(3,3)*M(1,1)-(M(3,1)*M(1,3)))/(M(3,3)*M(2,2
     . )*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1))
     . +M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(2
     . ,2)*M(1,3)))
      MINV(2,3)=(-(M(2,3)*M(1,1))+M(2,1)*M(1,3))/(M(3,3)*M(2,
     . 2)*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1)
     . )+M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(
     . 2,2)*M(1,3)))
      MINV(3,1)=(M(3,2)*M(2,1)-(M(3,1)*M(2,2)))/(M(3,3)*M(2,2
     . )*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1))
     . +M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(2
     . ,2)*M(1,3)))
      MINV(3,2)=(-(M(3,2)*M(1,1))+M(3,1)*M(1,2))/(M(3,3)*M(2,
     . 2)*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1)
     . )+M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(
     . 2,2)*M(1,3)))
      MINV(3,3)=(M(2,2)*M(1,1)-(M(2,1)*M(1,2)))/(M(3,3)*M(2,2
     . )*M(1,1)-(M(3,3)*M(2,1)*M(1,2))-(M(3,2)*M(2,3)*M(1,1))
     . +M(3,2)*M(2,1)*M(1,3)+M(3,1)*M(2,3)*M(1,2)-(M(3,1)*M(2
     . ,2)*M(1,3)))
      RETURN
      END
```

This is an example of a modular approach to code generation; separate subprogram templates are given in separate files. Furthermore, the template files are general; they can be used for matrices of any predetermined size. Therefore, we can easily generate different subprograms to handle matrices of different sizes from the same template files simply by assigning different values to n, and reloading the file init.red.

3.4 Template Processing and Generation of Type Declarations

In Section 2.11 we described the **GENDECS** flag. We explained that type declarations are not generated when this flag is turned off. Now that the concept of template processing has been explained, it is appropriate to continue our discussion of generation of type declarations.

When the **GENDECS** flag is off, type declaration information is not simply discarded — it is still maintained in the symbol table. Only the automatic extraction of this information in the form of declarations is disabled. When the **GENDECS** flag is turned off, all type information associated with a specific subprogram can be retrieved in the form of generated declarations by calling the **GENDECS** function with the subprogram name as argument. The template processor recognizes function and subroutine headings. It always keeps track of the name of the subprogram it is processing. Therefore, the declarations associated with a particular subprogram *subprogname* can be generated with a call to **GENDECS** as follows:

GENDECS *subprogname*$

By using the **GENDECS** flag and function together with the template processing facility, it is possible to have type information inserted into the symbol table during a first pass over a template file, and then to have it extracted during a second pass. Consider the following example in which the original template file is transformed into an intermediate template during the first pass, and then into the final file of FORTRAN code during the second pass:

Contents of file `junk.tem`:

```
;BEGIN;
MAXEXPPRINTLEN!* := 50$
OFF GENDECS$
```

```
;END;
      SUBROUTINE CALC(X,Y,Z,A,B,RES)
;BEGIN;
GENTRAN LITERAL ";BEGIN;", CR!*,
                "GENDECS CALC$", CR!*,
                 ";END;", CR!*$
;END;
      X=3.75
      Y=-10.2
      Z=16.473
;BEGIN;
GENTRAN
<<
    DECLARE X,Y,Z,A,B,RES : REAL;
    RES :=: (X + Y + Z)^3*(A + B)^2
>>$
;END;
      RETURN
      END
;BEGIN;
GENTRAN LITERAL ";END;", CR!*$
;END;
;END;
```

Invocation of the template processor on this file produces an intermediate template file:

```
1: GENTRANIN
1:     "junk.tem"
1: OUT "#junk.tem"$
```

Contents of file `#junk.tem`:

```
      SUBROUTINE CALC(X,Y,Z,A,B,RES)
;BEGIN;
GENDECS CALC$
;END;
      X=3.75
      Y=-10.2
      Z=16.473
      T0=A**2*X**3+3.0*A**2*X**2*Y
```

```
      T0=T0+3.0*A**2*X**2*Z+3.0*A**2*X*Y**2
      T0=T0+6.0*A**2*X*Y*Z+3.0*A**2*X*Z**2
      T0=T0+A**2*Y**3+3.0*A**2*Y**2*Z
      T0=T0+3.0*A**2*Y*Z**2+A**2*Z**3
      T0=T0+2.0*A*B*X**3+6.0*A*B*X**2*Y
      T0=T0+6.0*A*B*X**2*Z+6.0*A*B*X*Y**2
      T0=T0+12.0*A*B*X*Y*Z+6.0*A*B*X*Z**2
      T0=T0+2.0*A*B*Y**3+6.0*A*B*Y**2*Z
      T0=T0+6.0*A*B*Y*Z**2+2.0*A*B*Z**3
      T0=T0+B**2*X**3+3.0*B**2*X**2*Y
      T0=T0+3.0*B**2*X**2*Z+3.0*B**2*X*Y**2
      T0=T0+6.0*B**2*X*Y*Z+3.0*B**2*X*Z**2
      T0=T0+B**2*Y**3+3.0*B**2*Y**2*Z
      RES=T0+3.0*B**2*Y*Z**2+B**2*Z**3
      RETURN
      END
;END;
```

Another pass of the template processor produced the final file of FORTRAN code:

```
2: GENTRANIN
2:     "#junk.tem"
2: OUT "junk.f"$
```

Contents of file `junk.f`:

```
      SUBROUTINE CALC(X,Y,Z,A,B,RES)
      REAL X,Y,Z,A,B,RES,T0
      X=3.75
      Y=-10.2
      Z=16.473
      T0=A**2*X**3+3.0*A**2*X**2*Y
      T0=T0+3.0*A**2*X**2*Z+3.0*A**2*X*Y**2
      T0=T0+6.0*A**2*X*Y*Z+3.0*A**2*X*Z**2
      T0=T0+A**2*Y**3+3.0*A**2*Y**2*Z
      T0=T0+3.0*A**2*Y*Z**2+A**2*Z**3
      T0=T0+2.0*A*B*X**3+6.0*A*B*X**2*Y
      T0=T0+6.0*A*B*X**2*Z+6.0*A*B*X*Y**2
      T0=T0+12.0*A*B*X*Y*Z+6.0*A*B*X*Z**2
      T0=T0+2.0*A*B*Y**3+6.0*A*B*Y**2*Z
```

```
      T0=T0+6.0*A*B*Y*Z**2+2.0*A*B*Z**3
      T0=T0+B**2*X**3+3.0*B**2*X**2*Y
      T0=T0+3.0*B**2*X**2*Z+3.0*B**2*X*Y**2
      T0=T0+6.0*B**2*X*Y*Z+3.0*B**2*X*Z**2
      T0=T0+B**2*Y**3+3.0*B**2*Y**2*Z
      RES=T0+3.0*B**2*Y*Z**2+B**2*Z**3
      RETURN
      END
```

3.5 Referencing Subprogram and Parameter Names

In some code generation applications in which template processing is used, it is useful to be able to reference the names of the parameters given in the subprogram header. For this reason, the special symbols **!$1**, **!$2**, ..., **!$**$n$, where n is the number of parameters, can be used in computations and code generation commands in active parts of template files. Each of these symbols will be replaced by the corresponding parameter name when code is generated. In addition, the special symbol **!$0** will be replaced by the subprogram name. This is useful when FORTRAN or RATFOR functions are being generated. Finally, the special global variable **!$!#** is bound to the number of parameters in the subprogram header.

4 Output Redirection

Many examples given thus far in this manual have sent all generated code to the terminal screen. In actual code generation applications, however, code must be sent to a file which will be compiled at a later time. This section explains methods of redirecting code to a file as it is generated. Any number of output files can be open simultaneously, and generated code can be sent to any combination of these open files.

4.1 File Selection Commands

REDUCE provides the user with two file handling commands for output redirection: **OUT** and **SHUT**. The **OUT** command takes a single file name as argument and directs all REDUCE output to that file from then on, until

another **OUT** changes the output file, or **SHUT** closes it. Output can go to only one file at a time, although many can be open. If the file has previously been used for output during the current job and not **SHUT**, then the new output is appended onto the end of the file. Any existing file is erased before its first use for output in a job. To output on the terminal without closing the output file, the reserved file name **T** (for terminal) may be used.

The REDUCE **SHUT** command takes a list of names of files which have been previously opened via an **OUT** command and closes them. Most systems require this action by the user before he ends the REDUCE job; otherwise the output may be lost. If a file is **SHUT** and a further **OUT** command is issued for the same file, the file is erased before the new output is written. If it is the current output file that is **SHUT**, output will switch to the terminal.

These commands are suitable for most applications in which REDUCE output must be saved. However, they have two deficiencies when considered for use in code generation applications. First, they are inconvenient. **OUT** tells REDUCE to direct *all* output to a specified file. Thus in addition to output written as side effects of functions, returned values are also written to the file (unless the user is careful to terminate all statements and commands with a **$**, in which case only output produced by side effects is written). If code generation is to be accomplished interactively; i.e., if algebraic computations and code generation commands are interleaved, then **OUT** *filename***$** must be issued before every group of code generation requests, and **OUT T$** must be issued after every group. Secondly, the **OUT** command does not allow output to be sent to two or more files without reissuing the **OUT** with another file name. In an effort to remove these deficiencies and make the code generation commands flexible and easy to use, separate file handling commands are provided by GENTRAN which redirect generated code *only*.

The **GENTRANOUT** and **GENTRANSHUT** commands are identical to the REDUCE **OUT** and **SHUT** commands with the following exceptions:

- **GENTRANOUT** and **GENTRANSHUT** redirect *only* code which is printed as a side effect of GENTRAN commands.
- **GENTRANOUT** allows more than one file name to be given to indicate that generated code is to be sent to two or more files. (It is particularly convenient to be able to have generated code sent to the terminal screen and one or more file simultaneously.)

- **GENTRANOUT** does not automatically erase existing files; it prints a warning message on the terminal and asks the user whether the existing file should be erased or the whole command be aborted.

The next two subsections describe these commands in detail.

4.1.1 GENTRANOUT

Syntax:

GENTRANOUT *f1,f2,... ,fn;*

Arguments:

f1,f2,... ,fn is a list of one or more *f*'s, where each *f* is one of:

an atom	=	an output file
T	=	the terminal
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN

Side Effects:

GENTRAN maintains a list of files currently open for output by GENTRAN *only*. **GENTRANOUT** inserts each file name represented by *f1,f2,... ,fn* into that list and opens each one for output. It also resets the current output file(s) to be all files in *f1,f2,... ,fn*.

Returned Value:

GENTRANOUT returns the list of files represented by *f1,f2,... ,fn*; i.e., the current output file(s) after the command has been executed.

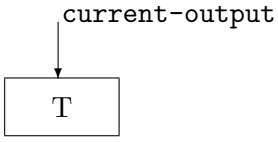
Diagnostic Messages:

```
*** OUTPUT FILE ALREADY EXISTS
    OVERWRITE FILE? (Y/N)

***** WRONG TYPE OF ARG
```

Example 14

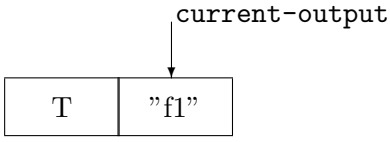
Output file list:




```
1: GENTRANOUT "f1";
```

```
"f1"
```

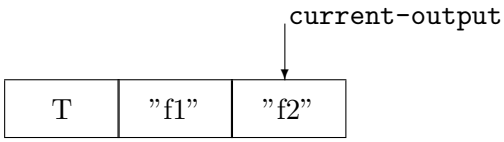
Output file list:




```
2: GENTRANOUT "f2";
```

```
"f2"
```

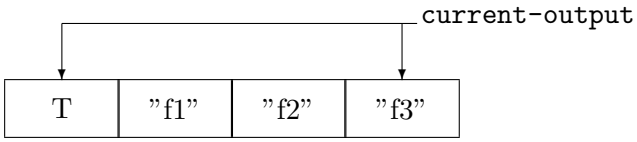
Output file list:




```
3: GENTRANOUT T,"f3";
```

```
{T,"f3"}
```

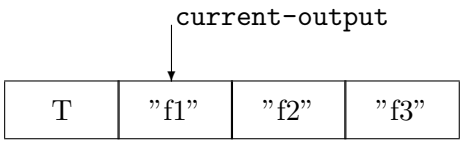
Output file list:




```
4: GENTRANOUT "f1";
```

```
"f1"
```

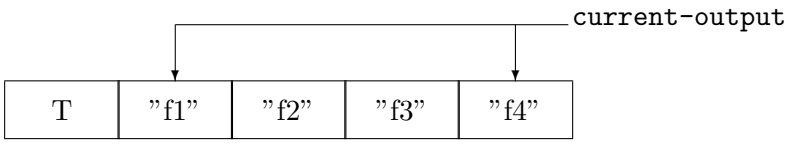
Output file list:




```
5: GENTRANOUT NIL,"f4";

{"f1","f4"}
```

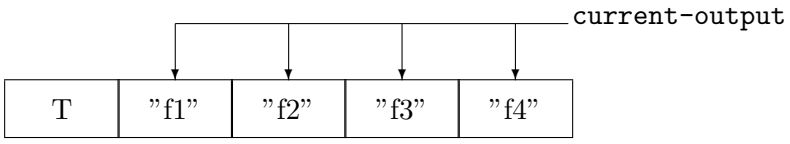
Output file list:




```
6: GENTRANOUT ALL!*;

{"f1","f2","f3","f4"}
```

Output file list:



4.1.2 GENTRANSHUT

Syntax:

GENTRANSHUT *f1,f2,... ,fn;*

Arguments:

f1,f2,... ,fn is a list of one or more *f*'s, where each *f* is one of:

an atom	=	an output file
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN

Side Effects:

GENTRANSHUT creates a list of file names from *f1,f2,...,fn*, deletes each from the output file list, and closes the corresponding files. If (all of) the current output file(s) are closed, then the current output file is reset to the terminal.

Returned Value:

GENTRANSHUT returns the current output file(s) after the command has been executed.

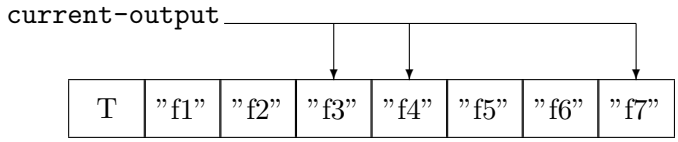
Diagnostic Messages:

```
*** FILE NOT OPEN FOR OUTPUT

***** WRONG TYPE OF ARG
```

Example 15

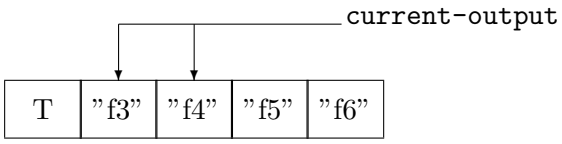
Output file list:




```
1: GENTRANSHUT "f1","f2","f7";

{"f3","f4"}
```

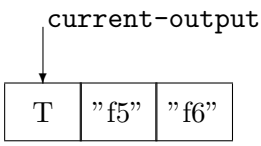
Output file list:




```
2: GENTRANSHUT NIL;

T
```

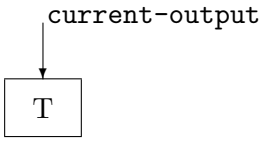
Output file list:



```
3: GENTRANSHUT ALL!*;

T
```

Output file list:



4.2 The Output File Stack

Section 4.1 explained the **GENTRANOUT** and **GENTRANSHUT** commands which are very similar to the REDUCE **OUT** and **SHUT** commands but redirect *only code generated as side effects of GENTRAN commands* to files. This section describes another pair of file handling commands provided by GENTRAN.

In some code generation applications it may be convenient to be able to send generated code to one (set of) file(s), then temporarily send code to another (set of) file(s), and later resume sending generated code to the first (set of) file(s). In other words, it is convenient to think of the output files as being arranged in a stack which can be pushed whenever new files are to be written to temporarily, and popped whenever previously written-to files are to be appended onto. **GENTRANPUSH** and **GENTRANPOP** enable the user to manipulate a stack of open output files in these ways.

GENTRANPUSH simply pushes a (set of) file(s) onto the stack and opens each one that is not already open for output. **GENTRANPOP** deletes the top-most occurrence of the given file(s) from the stack and closes each one that is no longer in the stack. The stack is initialized to one element: the terminal. This element is always on the bottom of the stack, and thus, is the default output file. The current output file is always the file(s) on top of the stack.

4.2.1 GENTRANPUSH

Syntax:

GENTRANPUSH *f1,f2,... ,fn;*

Arguments:

f1,f2,... ,fn is a list of one or more *f*'s, where each *f* is one of:

an atom	=	an output file
T	=	the terminal
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN

Side Effects:

GENTRANPUSH creates a list of file name(s) represented by *f1,f2,... ,fn* and pushes that list onto the output stack. Each file in the list that is not already open for output is opened at this time. The current output file is reset to this new element on the top of the stack.

Returned Value:

GENTRANPUSH returns the list of files represented by *f1,f2,... ,fn*; i.e., the current output file(s) after the command has been executed.

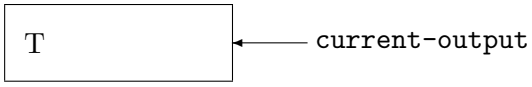
Diagnostic Messages:

```
*** OUTPUT FILE ALREADY EXISTS
    OVERWRITE FILE? (Y/N)

***** WRONG TYPE OF ARG
```

Example 16

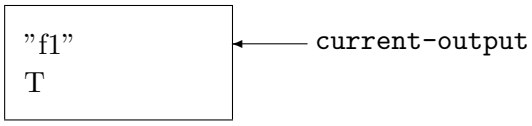
Output stack:




```
1: GENTRANPUSH "f1";

"f1"
```

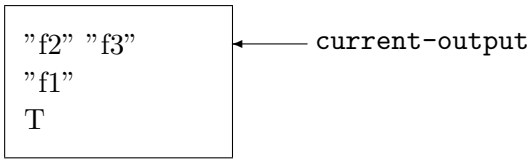
Output stack:




```
2: GENTRANPUSH "f2","f3";

{"f2","f3"}
```

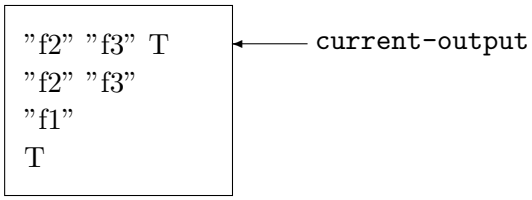
Output stack:




```
3: GENTRANPUSH NIL,T;

{"f2","f3",T}
```

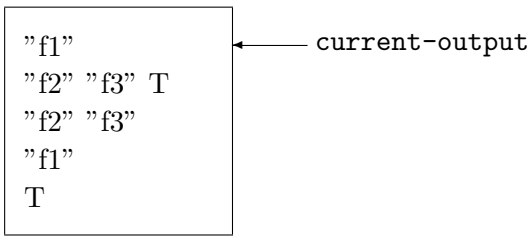
Output stack:




```
4: GENTRANPUSH "f1";

"f1"
```

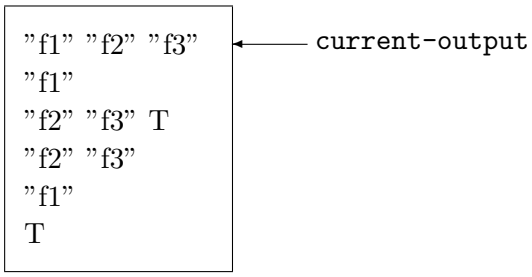
Output stack:




```
5: GENTRANPUSH ALL!*;

{"f1","f2","f3"}
```

Output stack:



4.2.2 GENTRANPOP

Syntax:

GENTRANPOP *f1,f2,... ,fn;*

Arguments:

f1,f2,... ,fn is a list of one or more *f*'s, where each *f* is one of:

an atom	=	an output file
T	=	the terminal
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN

Side Effects:

GENTRANPOP deletes the top-most occurrence of the single element containing the file name(s) represented by *f1,f2,... ,fn* from the output stack. Files whose names have been completely removed from the output stack are closed. The current output file is reset to the (new) element on the top of the output stack.

Returned Value:

GENTRANPOP returns the current output file(s) after this command has been executed.

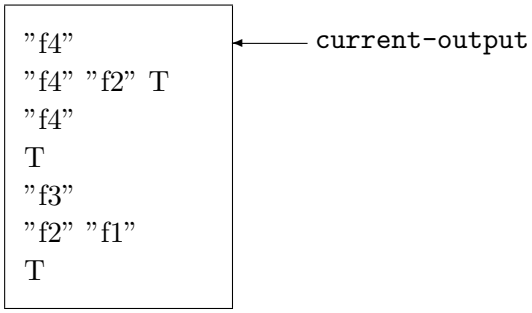
Diagnostic Messages:

```
*** FILE NOT OPEN FOR OUTPUT

***** WRONG TYPE OF ARG
```

Example 17

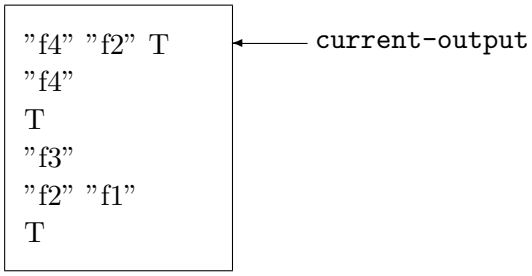
Output stack:




```
1: GENTRANPOP NIL;

{"f4","f2",T}
```

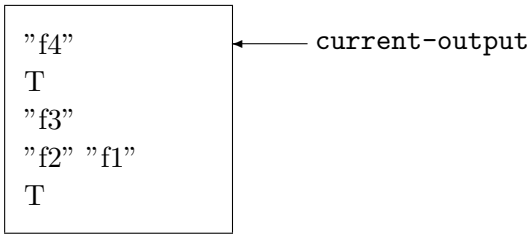
Output stack:




```
2: GENTRANPOP NIL;

"f4"
```

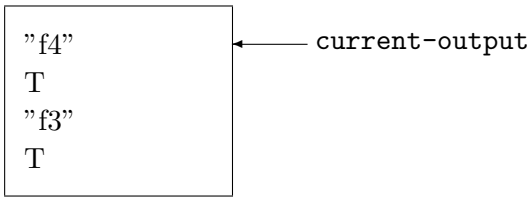
Output stack:




```
3: GENTRANPOP "f2","f1";

"f4"
```

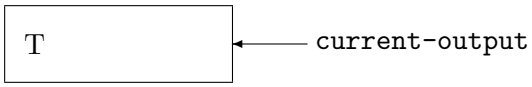
Output stack:




```
4: GENTRANPOP ALL!*;

T
```

Output stack:



4.3 Temporary Output Redirection

Sections 2.2 and 3.1 explain how to use the code generation and template processing commands. The syntax for these two commands is:

GENTRAN *stmt* [**OUT** *f1,f2,... ,fn*]*;*
and
GENTRANIN *f1,f2,... ,fm* [**OUT** *f1,f2,... ,fn*]*;*

The optional parts of these two commands can be used for *temporary* output redirection; they can be used when the current output file is to be temporarily reset, for this command only.

Thus the following two sequences of commands are equivalent:

```
10: GENTRANPUSH "f1",T$

11: GENTRAN ... $

12: GENTRANPOP NIL$
```

and

```
10: GENTRAN
10:    ...
10: OUT "f1",T$
```

5 Modification of the Code Generation Process

GENTRAN is designed to be flexible enough to be used in a variety of code generation applications. For this reason, several mode switches and variables are provided to enable the user to tailor the code generation process to meet his or her particular needs.

5.1 Mode Switches

The following GENTRAN mode switches can be turned on and off with the REDUCE **ON** and **OFF** commands.

DOUBLE

- When turned on, causes (where appropriate):
  - floating point numbers to be printed in double precision format;
  - intrinsic functions to be replaced by their double precision counterparts;
  - generated type declarations to be of double precision form.

  See also section 2.3 on page 8.
- default setting: off

GENDECS

- when turned on, allows type declarations to be generated automatically; otherwise, type information is stored in but not automatically retrieved from the symbol table. See also sections 2.5 on page 13, 2.7 on page 18, and 3.4 on page 41.
- default setting: on

GENTRANOPT

- when turned on, replaces each block of straightline code by an optimized sequence of assignments. The Code Optimizer takes a sequence of assignments and replaces common subexpressions with temporary variables. It returns the resulting assignment statements with common-subexpression-to-temporary-variable assignment statements preceding them
- default setting: off

GENTRANSEG

- when turned on, checks the print length of expressions and breaks those expressions that are longer than **MAXEXPPRINTLEN!*** down into subexpressions which are assigned to temporary variables. See also section 2.9 on page 21.
- default setting: on

GETDECS

- when on, causes:
  - the indices of loops to be declared integer;
  - objects without an explicit type declaration to be declared of the type given by the variable **DEFTYPE!***.

  See also section 2.6 on page 17.
- default setting: off

KEEPDECS

- when on, prevents declarations being removed from the symbol table when type declarations are generated.
- default: off

MAKECALLS

- when turned on, causes GENTRAN to translate functional expressions as subprogram calls.
- default setting: on

PERIOD

- when turned on, causes all integers to be printed out as floating point numbers except:
  - exponents;
  - variable subscripts;
  - index values in DO-type loops;
  - those which have been declared to be integers.
- default setting: on

5.2 Variables

Several global variables are provided in GENTRAN to enable the user to

- select the target language
- control expression segmentation
- change automatically generated variable names and statement numbers
- modify the code formatter

The following four subsections describe these variables[7].

5.2.1 Target Language Selection

GENTRANLANG!*

- target language (FORTRAN, RATFOR, PASCAL or C) See also section 2.1 on page 5.
- value type: atom
- default value: FORTRAN

5.2.2 Expression Segmentation Control

MAXEXPPRINTLEN!*

- value used to determine whether or not an expression should be segmented; maximum number of characters permitted in an expression in the target language (excluding spaces printed for formatting). See also section 2.9 on page 21.
- value type: integer
- default value: 800

5.2.3 Variable Names & Statement Numbers

TEMPVARNAME!*

- name used as prefix in generating temporary variable names. See also section 2.10 on page 23.
- value type: atom

[7]Note that when an atomic value (other than an integer) is assigned to a variable, that value must be quoted. For example, **GENTRANLANG!* := 'FORTRAN$** assigns the atom **FORTRAN** to the variable **GENTRANLANG!***.

- default value: T

TEMPVARNUM!*

- number appended to **TEMPVARNAME!*** to create a temporary variable name. If the temporary variable name resulting from appending **TEMPVARNUM!*** onto **TEMPVARNAME!*** has already been generated and still holds a useful value, then **TEMPVARNUM!*** is incremented and temporary variable names are compressed until one is found which was not previously generated or does not still hold a significant value. See also section 2.10 on page 23.
- value type: integer
- default value: 0

TEMPVARTYPE!*

- target language variable type (e.g., INTEGER, REAL!*8, FLOAT, etc) used as a default for automatically generated variables whose type cannot be determined otherwise. If **TEMPVARTYPE!*** is NIL, then generated temporary variables whose type cannot be determined are not automatically declared. See also section 2.10 on page 23.
- value type: atom
- default value: NIL

GENSTMTNUM!*

- number used when a statement number must be generated
- value type: integer
- default value: 25000

GENSTMTINCR!*

- number by which **GENSTMTNUM!*** is increased each time a new statement number is generated.
- value type: integer
- default value: 1

DEFTYPE!*

- default type for objects when the switch **GETDECS** is on. See also section 2.6 on page 17.

- value type: atom
- default value: real

5.2.4 Code Formatting

FORTCURRIND!*

- number of blank spaces printed at the beginning of each line of generated FORTRAN code beyond column 6
- value type: integer
- default value: 0

RATCURRIND!*

- number of blank spaces printed at the beginning of each line of generated RATFOR code.
- value type: integer
- default value: 0

CCURRIND!*

- number of blank spaces printed at the beginning of each line of generated C code.
- value type: integer
- default value: 0

PASCCURRIND!*

- number of blank spaces printed at the beginning of each line of generated PASCAL code.
- value type: integer
- default value: 0

TABLEN!*

- number of blank spaces printed for each new level of indentation.
- value type: integer
- default value: 4

FORTLINELEN!*

- maximum number of characters printed on each line of generated FORTRAN code.
- value type: integer
- default value: 72

RATLINELEN!*

- maximum number of characters printed on each line of generated RATFOR code.
- value type: integer
- default value: 80

CLINELEN!*

- maximum number of characters printed on each line of generated C code.
- value type: integer
- default value: 80

PASCLINELEN!*

- maximum number of characters printed on each line of generated PASCAL code.
- value type: integer
- default value: 70

MINFORTLINELEN!*

- minimum number of characters printed on each line of generated FORTRAN code after indentation.
- value type: integer
- default value: 40

MINRATLINELEN!*

- minimum number of characters printed on each line of generated RATFOR code after indentation.
- value type: integer
- default value: 40

MINCLINELEN!*

- minimum number of characters printed on each line of generated C code after indentation.

- value type: integer
- default value: 40

MINPASCLINELEN!*

- minimum number of characters printed on each line of generated PASCAL code after indentation.
- value type: integer
- default value: 40

6 Examples

Short examples have been given throughout this manual to illustrate usage of the GENTRAN commands. This section gives complete code generation examples.

6.1 Interactive Code Generation

Suppose we wish to generate a FORTRAN subprogram which can be used for computing the roots of a polynomial by Graeffe's Root-Squaring Method[8]. This method states that the roots x_i of a polynomial

$$P_n(x) = \sum_{i=0}^{n} a_i x^{n-i}$$

can be found by constructing the polynomial

$$P_n^*\left(x^2\right) = \left(a_0 x^n + a_2 x^{n-2} + \ldots\right)^2 - \left(a_1 x^{n-1} + a_3 x^{n-3} + \ldots\right)^2$$

with roots x_i^2 When read into REDUCE, the following file of REDUCE statements will place the coefficients of P_n^* into the list B for some user-entered value of n greater than zero.

Contents of file `graeffe.red`:[9]

[8] This is for instance convenient for ill-conditioned polynomials. More details are given in *Introduction to Numerical Analysis* by C. E. Froberg, Addison-Wesley Publishing Company, 1966.

[9] In accordance with section 2.5, the subscripts of A are I+1 instead of I.

```
OPERATOR A$
Q := FOR I := 0 STEP 2 UNTIL n   SUM (A(I+1)*X^(n-I))$
R := FOR I := 1 STEP 2 UNTIL n-1 SUM (A(I+1)*X^(n-I))$
P := Q^2 - R^2$
LET X^2 = Y$
B := COEFF(P,Y)$
END$
```

Now a numerical subprogram can be generated with assignment statements for the coefficients of P_n^* (now stored in list B in REDUCE). Since these coefficients are given in terms of the coefficients of P_n (i.e., operator A in REDUCE), the subprogram will need two parameters: A and B, each of which must be arrays of size n+1.

The following REDUCE session will create subroutine GRAEFF for a polynomial of degree n=10 and write it to file `graeffe.f`:

```
1: n := 10$

2: IN "graeffe.red"$

3: GENTRANLANG!* := 'FORTRAN$

4: ON DOUBLE$

5: GENTRAN
5: (
5:      PROCEDURE GRAEFF(A,B);
5:      BEGIN
5:      DECLARE
5:      <<
5:          GRAEFF : SUBROUTINE;
5:          A(11),B(11) : REAL
5:      >>;
5:      LITERAL
5:       "C",CR!*,
5:       "C",TAB!*,"GRAEFFE ROOT-SQUARING METHOD TO FIND",CR!*,
5:       "C",TAB!*,"ROOTS OF A POLYNOMIAL",CR!*,
5:       "C",CR!*;
5:      B(1) :=: PART (B,1);
5:      B(2) :=: PART (B,2);
5:      B(3) :=: PART (B,3);
5:      B(4) :=: PART (B,4);
```

```
5:      B(5) :=: PART (B,5);
5:      B(6) :=: PART (B,6);
5:      B(7) :=: PART (B,7);
5:      B(8) :=: PART (B,8);
5:      B(9) :=: PART (B,9);
5:      B(10) :=: PART (B,10);
5:      B(11) :=: PART (B,11)
5:      END
5: )
5: OUT "graeffe.f"$
```

Contents of file `graeffe.f`:

```
      SUBROUTINE GRAEFF(A,B)
      DOUBLE PRECISION A(11),B(11)
C
C     GRAEFFE ROOT-SQUARING METHOD TO FIND
C     ROOTS OF A POLYNOMIAL
C
      B(1)=A(11)**2
      B(2)=2.0D0*A(11)*A(9)-A(10)**2
      B(3)=2.0D0*A(11)*A(7)-(2.0D0*A(10)*A(8))+A(9)**2
      B(4)=2.0D0*A(11)*A(5)-(2.0D0*A(10)*A(6))+2.0D0*A(9)*A(7
     . )-A(8)**2
      B(5)=2.0D0*A(11)*A(3)-(2.0D0*A(10)*A(4))+2.0D0*A(9)*A(5
     . )-(2.0D0*A(8)*A(6))+A(7)**2
      B(6)=2.0D0*A(11)*A(1)-(2.0D0*A(10)*A(2))+2.0D0*A(9)*A(3
     . )-(2.0D0*A(8)*A(4))+2.0D0*A(7)*A(5)-A(6)**2
      B(7)=2.0D0*A(9)*A(1)-(2.0D0*A(8)*A(2))+2.0D0*A(7)*A(3)-
     . (2.0D0*A(6)*A(4))+A(5)**2
      B(8)=2.0D0*A(7)*A(1)-(2.0D0*A(6)*A(2))+2.0D0*A(5)*A(3)-
     . A(4)**2
      B(9)=2.0D0*A(5)*A(1)-(2.0D0*A(4)*A(2))+A(3)**2
      B(10)=2.0D0*A(3)*A(1)-A(2)**2
      B(11)=A(1)**2
      RETURN
      END
```

6.2 Code Generation, Segmentation & Temporary Variables

The following 3 x 3 inertia matrix M was derived in the course of some research [10]:

$$
\begin{aligned}
M(1,1) &= 18*\cos(q_3)*\cos(q_2)*m_{30}*p^2 - \sin^2(q_3)*j_{30}y + \sin^2(q_3) \\
&\quad *j_{30}z - 9*\sin^2(q_3)*m_{30}*p^2 + j_{10}y + j_{30}y + m_{10}*p^2 + \\
&\quad 18*m_{30}*p^2 \\
M(1,2) &= 9*\cos(q_3)*\cos(q_2)*m_{30}*p^2 - \sin^2(q_3)*j_{30}y + \sin^2(q_3) \\
&\quad *j_{30}z - 9*\sin^2(q_3)*m_{30}*p^2 + j_{30}y + 9*m_{30}*p^2 \\
M(2,1) &= M(1,2) \\
M(1,3) &= -9*\sin(q_3)*\sin(q_2)*m_{30}*p^2 \\
M(3,1) &= M(1,3) \\
M(2,2) &= -\sin^2(q_3)*j_{30}y + \sin^2(q_3)*j_{30}z - 9*\sin^2(q_3)*m_{30}*p^2 \\
&\quad +j_{30}y + 9*m_{30}*p^2 \\
M(2,3) &= 0 \\
M(3,2) &= M(2,3) \\
M(3,3) &= 9*m_{30}*p^2 + j_{30}x
\end{aligned}
$$

We know M is symmetric. We wish to generate numerical code to compute values for M and its inverse matrix.

6.2.1 Code Generation

Generating code for matrix M and its inverse matrix is straightforward. We can simply generate an assignment statement for each element of M, compute the inverse matrix MIV, and generate an assignment statement for each element of MIV. Since we know M is symmetric, we know that MIV will also be symmetric. To avoid duplicate computations, we will not generate assignments for elements below the main diagonals of these matrices. Instead, we will copy elements across the main diagonal by generating nested loops. The following REDUCE session will write to the file `m1.f`:

[10] For details see: Bos, A. M. and M. J. L. Tiernego. "Formula Manipulation in the Bond Graph Modelling and Simulation of Large Mechanical Systems", *Journal of the Franklin Institute* , Pergamon Press Ltd., Vol. 319, No. 1/2, pp. 51-65, January/February 1985.

```
1: IN "m.red"$ % Initialize M

2: GENTRANOUT "m1.f"$

3: GENTRANLANG!* := 'FORTRAN$

4: ON DOUBLE$

5: FOR J := 1 : 3 DO
5:      FOR K := J : 3 DO
5:          GENTRAN M(J,K) ::=: M(J,K)$

6: MIV := M^(-1)$

7: FOR J := 1 : 3 DO
7:      FOR K := J : 3 DO
7:          GENTRAN MIV(J,K) ::=: MIV(J,K)$

8: GENTRAN
8:      FOR J := 1 : 3 DO
8:          FOR K := J+1 : 3 DO
8:          <<
8:              M(K,J) := M(J,K);
8:              MIV(K,J) := MIV(J,K)
8:          >>$

9: GENTRANSHUT "m1.f"$
```

The contents of `m1.f` are reproduced in 10 on page 111.

This code was generated with the segmentation facility turned off. However, most FORTRAN compilers cannot handle statements more than 20 lines long. The next section shows how to generate segmented assignments.

6.2.2 Segmentation

Large arithmetic expressions can be broken into pieces of manageable size with the expression segmentation facility. The following REDUCE session will write segmented assignment statements to the file `m2.f`. Large arith-

metic expressions will be broken into subexpressions of approximately 300 characters in length.

```
1: IN "m.red"$ % Initialize M

2: GENTRANOUT "m2.f"$

3: ON DOUBLE$

4: ON GENTRANSEG$

5: MAXEXPPRINTLEN!* := 300$

6: FOR J := 1 : 3 DO
6:      FOR K := J : 3 DO
6:          GENTRAN M(J,K) ::=: M(J,K)$

7: MIV := M^(-1)$

8: FOR J := 1 : 3 DO
8:      FOR K := J : 3 DO
8:          GENTRAN MIV(J,K) ::=: MIV(J,K)$

9: GENTRAN
9:      FOR J := 1 : 3 DO
9:          FOR K := J+1 : 3 DO
9:          <<
9:              M(K,J) := M(J,K);
9:              MIV(K,J) := MIV(J,K)
9:          >>$

10: GENTRANSHUT "m2.f"$
```

The contents of file `m2.f` are reproduced in 10 on page 111.

6.2.3 Generation of Temporary Variables to Suppress Simplification

We can dramatically improve the efficiency of the code generated in sections 6.2.1 on page 65 and 6.2.2 on page 66 by replacing expressions by temporary variables before computing the inverse matrix. This effectively suppresses simplification; these expressions will not be substituted into later computations. We will replace each non-zero element of the REDUCE matrix M by a generated variable name, and generate a numerical assignment statement to reflect that substitution in the numerical program being generated.

The following REDUCE session will write to the file `m3.f`:

```
1: in "m.red"$ % Initialize M

2: GENTRANOUT "m3.f"$

3: GENTRANLANG!* := 'FORTRAN$

4: ON DOUBLE$

5: FOR J := 1 : 3 DO
5:      FOR K := J : 3 DO
5:          GENTRAN M(J,K) ::=: M(J,K)$

6: SHARE VAR$

7: FOR J := 1 : 3 DO
7:      FOR K := J : 3 DO
7:          IF M(J,K) NEQ 0 THEN
7:          <<
7:              VAR := TEMPVAR(NIL)$
7:              MARKVAR VAR$
7:              M(J,K) := VAR$
7:              M(K,J) := VAR$
7:              GENTRAN
7:                  EVAL(VAR) := M(EVAL(J),EVAL(K))
7:          >>$
```

```
8: COMMENT ** Contents of matrix M: **$

9: M;

[T0  T1  T2]
[          ]
[T1  T3  0 ]
[          ]
[T2  0   T4]


10: MIV := M^(-1)$

11: FOR J := 1 : 3 DO
11:      FOR K := J : 3 DO
11:          GENTRAN MIV(J,K) ::=: MIV(J,K)$

12: GENTRAN
12:      FOR J := 1 : 3 DO
12:          FOR K := J+1 : 3 DO
12:          <<
12:              M(K,J) := M(J,K);
12:              MIV(K,J) := MIV(J,K)
12:          >>$

13: GENTRANSHUT "m3.f"$
```

Contents of file m3.f:

```
      M(1,1)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(DSIN(DBLE(
     . Q3))**2*Y*J30)+DSIN(DBLE(Q3))**2*J30Z+18.0D0*DCOS(DBLE
     . (Q3))*DCOS(DBLE(Q2))*P**2*M30+18.0D0*P**2*M30+P**2*M10
     . +J30Y+J10Y
      M(1,2)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(DSIN(DBLE(
     . Q3))**2*J30Y)+DSIN(DBLE(Q3))**2*J30Z+9.0D0*DCOS(DBLE(
     . Q3))*DCOS(DBLE(Q2))*P**2*M30+9.0D0*P**2*M30+J30Y
      M(1,3)=-(9.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*P**2*M30)
      M(2,2)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(DSIN(DBLE(
     . Q3))**2*J30Y)+DSIN(DBLE(Q3))**2*J30Z+9.0D0*P**2*M30+
     . J30Y
```

```
      M(2,3)=0.0D0
      M(3,3)=9.0D0*P**2*M30+J30X
      T0=M(1,1)
      T1=M(1,2)
      T2=M(1,3)
      T3=M(2,2)
      T4=M(3,3)
      MIV(1,1)=-(T4*T3)/(T4*T1**2-(T4*T3*T0)+T2**2*T3)
      MIV(1,2)=(T4*T1)/(T4*T1**2-(T4*T3*T0)+T2**2*T3)
      MIV(1,3)=(T2*T3)/(T4*T1**2-(T4*T3*T0)+T2**2*T3)
      MIV(2,2)=(-(T4*T0)+T2**2)/(T4*T1**2-(T4*T3*T0)+T2**2*
     . T3)
      MIV(2,3)=-(T1*T2)/(T4*T1**2-(T4*T3*T0)+T2**2*T3)
      MIV(3,3)=(T1**2-(T3*T0))/(T4*T1**2-(T4*T3*T0)+T2**2*T3)
      DO 25009 J=1,3
          DO 25010 K=J+1,3
              M(K,J)=M(J,K)
              MIV(K,J)=MIV(J,K)
25010     CONTINUE
25009 CONTINUE
```

6.3 Template Processing

Circuit simulation plays a vital role in computer hardware development. A recent paper[11] describes the design of an Automatic Circuitry Code Generator (ACCG), which generates circuit simulation programs based on user-supplied circuit specifications. The actual code generator consists of a series of REDUCE **WRITE** statements, each of which writes one line of FORTRAN code.

This section presents an alternative implementation for the ACCG which uses GENTRAN's template processor to generate code. Template processing is a much more natural method of code generation than the REDUCE **WRITE** statement method.

First we will put all REDUCE calculations into two files: `rk.red` and `ham.red`.

[11] Loe, K. F., N. Ohsawa, and E. Goto. "Design of an Automatic Circuitry Code Generator (ACCG)", *RSYMSAC Proceedings*, Wako-shi, Saitama, Japan. 1984.

Contents of file `rk.red`:[12]

```
COMMENT  -- RUNGE-KUTTA METHOD --$
PROCEDURE RUNGEKUTTA(P1, P2, P, Q, TT);
BEGIN
SCALAR K11,K12,K21,K22,K31,K32,K41,K42;
K11 := HH*P1;
K12 := HH*P2;
K21 := HH*SUB(TT=TT+HH/2, P=P+K11/2, Q=Q+K12/2, P1);
K22 := HH*SUB(TT=TT+HH/2, P=P+K11/2, Q=Q+K12/2, P2);
K31 := HH*SUB(TT=TT+HH/2, P=P+K21/2, Q=Q+K22/2, P1);
K32 := HH*SUB(TT=TT+HH/2, P=P+K21/2, Q=Q+K22/2, P2);
K41 := HH*SUB(TT=TT+HH,   P=P+K31,   Q=Q+K32,   P1);
K42 := HH*SUB(TT=TT+HH,   P=P+K31,   Q=Q+K32,   P2);
PN := P + (K11 + 2*K21 + 2*K31 + K41)/6;
QN := Q + (K12 + 2*K22 + 2*K32 + K42)/6
END$
END$
```

Contents of file `ham.red`:

```
COMMENT  -- HAMILTONIAN CALCULATION --$
DIFQ := DF(H,P)$
DIFP := -DF(H,Q) - SUB(QDOT=P/M, DF(D,QDOT))$
RUNGEKUTTA(DIFP, DIFQ, P, Q, TT)$
END$
```

Next we will create a template file with an outline of the target FORTRAN program and GENTRAN commands.

Contents of file `runge.tem`:

```
      PROGRAM RUNGE
      IMPLICIT DOUBLE PRECISION (K,M)
C
C  INPUT
C
      WRITE(6,*) 'INITIAL VALUE OF P'
```

[12] Line 11 of procedure RUNGEKUTTA was changed from

```
K41 := HH*SUB(TT=TT+HH, P=P+K31, Q=Q+K32, P2);
```

as given in (Loe84), to

```
K42 := HH*SUB(TT=TT+HH, P=P+K31, Q=Q+K32, P2);
```

```
      READ(5,*) P
      WRITE(6,*) ' P = ', P
      WRITE(6,*) 'INITIAL VALUE OF Q'
      READ(5,*) Q
      WRITE(6,*) ' Q = ', Q
      WRITE(6,*) 'VALUE OF M'
      READ(5,*) M
      WRITE(6,*) ' M = ', M
      WRITE(6,*) 'VALUE OF K0'
      READ(5,*) K0
      WRITE(6,*) ' K0 = ', K0
      WRITE(6,*) 'VALUE OF B'
      READ(5,*) B
      WRITE(6,*) ' B = ', B
      WRITE(6,*) 'STEP SIZE OF T'
      READ(5,*) HH
      WRITE(6,*) ' STEP SIZE OF T = ', HH
      WRITE(6,*) 'FINAL VALUE OF T'
      READ(5,*) TP
      WRITE(6,*) ' FINAL VALUE OF T = ', TP
C
C  INITIALIZATION
C
      TT=0.0D0
;BEGIN;
      GENTRAN
        LITERAL
          TAB!*, "WRITE(9,*) ' H = ", EVAL(H), "'", CR!*,
          TAB!*, "WRITE(9,*) ' D = ", EVAL(D), "'", CR!*$
;END;
      WRITE(9,901) C
901   FORMAT(' C= ',D20.10)
      WRITE(9,910) TT, Q, P
910   FORMAT(' '3D20.10)
C
C  LOOP
C
;BEGIN;
```

```
      GENTRAN
          REPEAT
          <<
              PN :=: PN;
              Q  :=: QN;
              P  := PN;
              TT := TT + HH;
              LITERAL
                TAB!*, "WRITE(9,910) TT, QQ, P", CR!*
          >>
          UNTIL TT >= TF$
;END;
      STOP
      END
;END;
```

Now we can generate a circuit simulation program simply by starting a REDUCE session and following three steps:

1. Enter circuit specifications.
2. Perform calculations.
3. Call the GENTRAN template processor.

For example, the following REDUCE session will write a simulation program to the file `runge.f`:

```
1: COMMENT  -- INPUT --$

2: K := 1/(2*M)*P^2$    % kinetic energy

3: U := K0/2*Q^2$       % potential energy

4: D := B/2*QDOT$       % dissipating function

5: H := K + U$          % hamiltonian

6: COMMENT  -- CALCULATIONS --$

7: IN "rk.red", "ham.red"$
```

```
8: COMMENT  -- FORTRAN CODE GENERATION --$

9: GENTRANLANG!* := 'FORTRAN$

10: ON DOUBLE$

11: GENTRANIN "runge.tem" OUT "runge.f"$
```

Contents of file `runge.f`:

```
      PROGRAM RUNGE
      IMPLICIT DOUBLE PRECISION (K,M)
C
C  INPUT
C
      WRITE(6,*) 'INITIAL VALUE OF P'
      READ(5,*) P
      WRITE(6,*) ' P = ', P
      WRITE(6,*) 'INITIAL VALUE OF Q'
      READ(5,*) Q
      WRITE(6,*) ' Q = ', Q
      WRITE(6,*) 'VALUE OF M'
      READ(5,*) M
      WRITE(6,*) ' M = ', M
      WRITE(6,*) 'VALUE OF K0'
      READ(5,*) K0
      WRITE(6,*) ' K0 = ', K0
      WRITE(6,*) 'VALUE OF B'
      READ(5,*) B
      WRITE(6,*) ' B = ', B
      WRITE(6,*) 'STEP SIZE OF T'
      READ(5,*) HH
      WRITE(6,*) ' STEP SIZE OF T = ', HH
      WRITE(6,*) 'FINAL VALUE OF T'
      READ(5,*) TP
      WRITE(6,*) ' FINAL VALUE OF T = ', TP
C
C  INITIALIZATION
C
      TT=0.0D0
```

```
      WRITE(9,*) ' H = (M*Q**2*K0+P**2)/(2.0D0*M)'
      WRITE(9,*) ' D = (B*QDOT)/2.0D0'
      WRITE(9,901) C
901   FORMAT(' C= ',D20.10)
      WRITE(9,910) TT, Q, P
910   FORMAT(' '3D20.10)
C
C  LOOP
C
25001 CONTINUE
          PN=(-(12.0D0*B*M**2*HH)+2.0D0*B*M*K0*HH**3+24.0D0*
     .     M**2*P-(24.0D0*M**2*Q*K0*HH)-(12.0D0*M*P*K0*HH**2)
     .     +4.0D0*M*Q*K0**2*HH**3+P*K0**2*HH**4)/(24.0D0*M**2
     .     )
          Q=(-(12.0D0*B*M*HH**2)+B*K0*HH**4+48.0D0*M**2*Q+
     .     48.0D0*M*P*HH-(24.0D0*M*Q*K0*HH**2)-(8.0D0*P*K0*HH
     .     **3)+2.0D0*Q*K0**2*HH**4)/(48.0D0*M**2)
          P=PN
          TT=TT+HH
          WRITE(9,910) TT, QQ, P
      IF (.NOT.TT.GE.TF) GOTO 25001
      STOP
      END
```

7 Symbolic Mode Functions

Thus far in this manual, commands have been presented which are meant to be used primarily in the algebraic mode of REDUCE. These commands are designed to be used interactively. However, many code generation applications require code to be generated under program control[13]. In these applications, it is generally more convenient to generate code from (computed) prefix forms. Therefore, GENTRAN provides code generation and file handling functions designed specifically to be used in the symbolic mode of REDUCE. This section presents the symbolic functions which are analogous to the code generation, template processing, and output file handling commands presented in sections 2, 3, and 4.

[13] [**?**] contains one such example.

7.1 Code Generation and Translation

Sections 2.2 through 2.8 describe interactive commands and functions which generate and translate code, declare variables to be of specific types, and insert literal strings of characters into the stream of generated code. This section describes analogous symbolic mode code generation functions.

7.1.1 Translation of Prefix Forms

In algebraic mode, the **GENTRAN** command translates algorithmic specifications supplied in the form of REDUCE statements into numerical code. Similarly, the symbolic function **SYM!-GENTRAN** translates algorithmic specifications supplied in the form of REDUCE prefix forms into numerical code.

Syntax:

SYM!-GENTRAN *form*;

Function Type:

expr

Argument:

form is any LISP prefix form that evaluates to a REDUCE prefix form that can be translated by GENTRAN into the target language[14]. *form* may contain any number of occurrences of the special forms **EVAL**, **LSETQ**, **RSETQ**, **LRSETQ**, **DECLARE**, and **LITERAL** (see sections 7.1.2 through 7.1.3 on pages 78–81).

Side Effects:

SYM!-GENTRAN translates *form* into formatted code in the target language and writes it to the file(s) currently selected for output.

Returned Value:

SYM!-GENTRAN returns the name(s) of the file(s) to which code was written. If code was written to one file, the returned

[14] See 8 on page 93 for a complete listing of REDUCE prefix forms that can be translated.

value is an atom; otherwise, it is a list.

Diagnostic Messages:

```
*** OUTPUT FILE ALREADY EXISTS

   OVERWRITE FILE? (Y/N)

***** WRONG TYPE OF ARG
```

exp

```
***** CANNOT BE TRANSLATED
```

Example 18

```
1: SYMBOLIC$

2: GENTRANLANG!* := 'FORTRAN$

3:   SYM!-GENTRAN '(FOR I (1 1 n) DO (SETQ (V I) 0))$

     DO 25001 I=1,N
          V(I)=0.0
25001 CONTINUE

4: GENTRANLANG!* := 'RATFOR$

5: SYM!-GENTRAN '(FOR I (1 1 N) DO
5:                    (FOR J ((PLUS I 1) 1 N) DO
5:                         (PROGN
5:                            (SETQ (X J I) (X I J))
5:                            (SETQ (Y J I) (Y I J)))))$

DO I=1,N
   DO J=I+1,N
        {
            X(J,I)=X(I,J)
            Y(J,I)=Y(I,J)
        }

6: GENTRANLANG!* := 'C$
```

```
7: SYM!-GENTRAN '(SETQ P (FOR I (1 1 N) PRODUCT I))$

{
    P=1;
    for (I=1;I<=N;++I)
        P*=I;
}

8:  GENTRANLANG!* := 'PASCAL$

9: SYM!-GENTRAN '(SETQ C
9:     (COND ((LESSP A B) A) (T B)))$
IF A<B THEN
    C:=A;
ELSE
    C:=B;
```

7.1.2 Code Generation

Sections 2.4.1 through 2.4.4 on pages 10–12 described the special functions and operators **EVAL**, **::=**, **:=:**, and **::=:** that could be included in arguments to the **GENTRAN** command to indicate that parts of those arguments were to be given to REDUCE FOR Evaluation prior to translation. This section describes the analogous functions that can be supplied in prefix form to the **SYM!-GENTRAN** function.

The following special forms may be interleaved arbitrarily in forms supplied as arguments to **SYM!-GENTRAN** to specify partial evaluation: **EVAL**, **LSETQ**, **RSETQ**, and **LRSETQ**. Sections 7.1.2 through 7.1.2 describe these forms. Then section 7.1.2 through 7.1.2 present examples of the usage of these forms for evaluation of expressions in both symbolic and algebraic modes.

The EVAL Form Syntax:

(EVAL *form*)

Argument:

form is any LISP prefix form that evaluates to a REDUCE prefix form that can be translated by GENTRAN into the target language.

The LSETQ Form Syntax:

(LSETQ *svar exp*)

Arguments:

svar is a subscripted variable in LISP prefix form. Its subscripts must evaluate to REDUCE prefix forms that can be translated into the target language. *exp* is any REDUCE expression in prefix form that can be translated by GENTRAN.

The RSETQ Form Syntax:

(RSETQ *var exp*)

Arguments:

var is a variable in REDUCE prefix form. *exp* is a LISP prefix form which evaluates to a translatable REDUCE prefix form.

The LRSETQ Form Syntax:

(LRSETQ *svar exp*)

Arguments:

svar is a subscripted variable in LISP prefix form with subscripts that evaluate to REDUCE prefix forms that can be translated by GENTRAN. *exp* is a LISP prefix form that evaluates to a translatable REDUCE prefix form.

Symbolic Mode Evaluation The symbolic mode evaluation forms that have just been described are analogous to their algebraic mode counterparts, except that by default, they evaluate their argument(s) in symbolic mode. The following is an example of evaluation of subscripts in symbolic mode:

Example 19

```
1: SYMBOLIC$

2: FOR i:=1:2 DO
2:     FOR j:=1:2 DO
2:         SYM!-GENTRAN '(LSETQ (M i j) 0)$

     M(1,1)=0.0
     M(1,2)=0.0
     M(2,1)=0.0
     M(2,2)=0.0
```

Algebraic Mode Evaluation As we have just seen, the symbolic mode evaluation forms evaluate their argument(s) in symbolic mode. This default evaluation mode can be overridden by explicitly requesting evaluation in algebraic mode with the REDUCE **AEVAL** function.

Example 20

```
1: ALGEBRAIC$

2: F := 2*x^2 - 5*X + 6$

3: SYMBOLIC$

4: SYM!-GENTRAN '(SETQ Q (QUOTIENT
4:                            (EVAL (AEVAL 'F))
4:                            (EVAL (AEVAL '(DF F X)))))$

     Q=(2.0*X**2-5.0*X+6.0)/(4.0*X-5.0)

5: ALGEBRAIC$

6: M := MAT(( A,   0,  -1,  1),
6:          ( 0, B^2,   0,  1),
6:          (-1,   B, B*C,  0),
6:          ( 1,   0,  -C, -D))$

7: SYMBOLIC$
```

```
8: FOR i:=1:4 DO
8:      SYM!-GENTRAN '(LRSETQ (M i i)
                      (AEVAL (MKQUOTE (LIST 'M i i))))$

     M(1,1)=A
     M(2,2)=B**2
     M(3,3)=B*C
     M(4,4)=-D
```

SHAREd Variables The REDUCE **SHARE** command enables variables to be shared between algebraic and symbolic modes. Thus, we can derive an expression in algebraic mode, assign it to a shared variable, and then access the value of that variable to generate code from symbolic mode.

Example:

```
1: ALGEBRAIC$

2: SHARE dfx1$

3: dfx1 := DF(X**4 - X**3 + 2*X**2 + 1, X)$

4: SYMBOLIC$

5: SYM!-GENTRAN '(RSETQ DERIV dfx1)$
      DERIV=4.0*X**3-(3.0*X**2)+4.0*X
```

7.1.3 Special Translatable Forms

Sections 2.5 through 2.8 described special functions that could be used to declare variable types and insert literal strings of characters into generated code. This section contains explanations of analogous prefix forms for usage in symbolic mode.

Explicit Type Declarations A similar form of the algebraic mode **DECLARE** function is provided in symbolic mode:

Syntax:

(DECLARE (*type1 v1 v2 ... vn1*)
(*type2 v1 v2 ... vn2*)
:
(*typen v1 v2 ... vnn*))

Arguments:

Each *v1 v2 ... vn* is a sequence of one or more variables (optionally subscripted to indicate array dimensions – in prefix form), or variable ranges (two letters concatenated together with "-" in between). *v*s are not evaluated unless given as arguments to **EVAL**.

Each *type* is a variable type in the target language. Each must be an atom, optionally concatenated to the atom **IMPLICIT!** (note the trailing space). *type*s are not evaluated unless given as arguments to **EVAL**.

Side Effect:

Entries are placed in the symbol table for each variable or variable range declared in the call to this function. The function call itself is removed from the statement group being translated. Then after translation, type declarations are generated from these symbol table entries before the resulting executable statements are printed.

Example 21

```
1: SYMBOLIC$

2: GENTRANLANG!* := 'FORTRAN$

3: SYM!-GENTRAN
3:    '(PROGN
3:        (DECLARE (IMPLICIT! REAL!*8 A!-H O!-Z)
3:                 (INTEGER (M 4 4)))
3:        (FOR I (1 1 4) DO
3:             (FOR J (1 1 4) DO
3:                  (COND ((EQUAL I J) (SETQ (M I J) 1))
3:                        (T (SETQ (M I J) 0)))))
3:        (DECLARE (INTEGER I J))
      :
```

```
      :
3:     )$

      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER M(4,4),I,J
      DO 25001 I=1,4
           DO 25002 J=1,4
               IF (I.EQ.J) THEN
                   M(I,J)=1
               ELSE
                   M(I,J)=0
               ENDIF
25002      CONTINUE
25001 CONTINUE
      :
      :

4: GENTRANLANG!* := 'RATFOR$

5: SYM!-GENTRAN
5:    '(PROCEDURE FAC NIL EXPR (N)
5:        (BLOCK ()
5:               (DECLARE (FUNCTION FAC)
5:                        (INTEGER FAC N))
5:               (SETQ F (FOR I (1 1 N) PRODUCT I))
5:               (DECLARE (INTEGER F I))
5:               (RETURN F)))$

INTEGER FUNCTION FAC(N)
INTEGER N,F,I
{
    F=1
    DO I=1,N
        F=F*I
}
RETURN(F)
END

6: GENTRANLANG!* := 'C$
```

```
7: SYM!-GENTRAN
7:    '(PROCEDURE FAC NIL EXPR (N)
7:        (BLOCK ()
7:               (DECLARE (INTEGER FAC N I F))
7:               (SETQ F (FOR I (1 1 N) PRODUCT I))
7:               (RETURN F)))$

int FAC(N)
int N;
{
    int I,F;
    {
        F=1;
        for (I=1;I<=N;++I)
            F*=I;
    }
    return(F);
}

8:  GENTRANLANG!* := 'PASCAL$

9:  SYM!-GENTRAN
9:     '(PROCEDURE FAC NIL EXPR (N)
9:         (BLOCK ()
9:                (DECLARE (INTEGER FAC N I F))
9:                (SETQ F (FOR I (1 1 N) PRODUCT I))
9:                (RETURN F)))$
FUNCTION FAC(N:INTEGER):INTEGER;
LABEL
    99999;
VAR
    I,F: INTEGER;
BEGIN
    BEGIN
        F:=1;
        FOR I:=1 TO N DO
            F:=F*I
    END;
```

```
    BEGIN
        FAC:=F;
        GOTO 99999{RETURN}
    END;
99999:
END;
```

Comments and Literal Strings A form similar to the algebraic mode **LITERAL** function is provided in symbolic mode:

Syntax:

(**LITERAL** *arg1 arg2 ... argn*)

Arguments:

arg1 arg2 ... argn is an argument sequence containing one or more *arg*s, where each *arg* either is, or evaluates to, an atom. The atoms **TAB!*** and **CR!*** have special meanings. *arg*s are not evaluated unless given as arguments to **EVAL**.

Side Effect:

This form is replaced by the character sequence resulting from concatenation of the given atoms. Double quotes are stripped from all string type *arg*s, and the reserved atoms **TAB!*** and **CR!*** are replaced by a tab to the current level of indentation, and an end-of-line character, respectively.

Example 22

```
1: SYMBOLIC$

2: GENTRANLANG!* := 'FORTRAN$

3: N := 100$

4: SYM!-GENTRAN
4:  '(PROGN
4:    (LITERAL C TAB!* "--THIS IS A FORTRAN COMMENT--"
4:             CR!* C CR!*)
```

```
4:    (LITERAL TAB!* "DATA N/" (EVAL N) "/" CR!*))$

C     --THIS IS A FORTRAN COMMENT--
C
      DATA N/100/

5: GENTRANLANG!* := 'RATFOR$

6: SYM!-GENTRAN
6:  '(FOR I (1 1 N) DO
6:      (PROGN
6:       (LITERAL TAB!* "# THIS IS A RATFOR COMMENT" CR!*)
6:       (LITERAL TAB!* "WRITE(6,10) (M(I,J),J=1,N)" CR!*
6:         10 TAB!* "FORMAT(1X,10(I5,3X))" CR!*)))$

DO I=1,N
   {
        # THIS IS A RATFOR COMMENT
        WRITE(6,10) (M(I,J),J=1,N)
10      FORMAT(1X,10(I5,3X))
   }

7: GENTRANLANG!* := 'C$

8: SYM!-GENTRAN
8:  '(PROGN
8:    (SETQ X 0)
8:    (LITERAL "/* THIS IS A" CR!* "
8:             C COMMENT */" CR!*))$

{
    X=0.0;
/* THIS IS A
   C COMMENT */
}

9: GENTRANLANG!* := 'PASCAL$

10: SYM!-GENTRAN
```

```
10:  '(PROGN
10:     (SETQ X (SIN Y))
10:     (LITERAL "{ THIS IS A PASCAL COMMENT }" CR!*))$
BEGIN
    X:=SIN(Y)
{ THIS IS A PASCAL COMMENT }
END;
```

7.2 Template Processing

The template processor can be invoked from either algebraic or symbolic mode. Section 3.1 described the algebraic mode command. This section describes the analogous symbolic mode function.

Syntax:

SYM!-GENTRANIN *list-of-fnames*;

Function Type:

expr

Argument:

list-of-fnames evaluates to a LISP list containing one or more *fname*s, where each *fname* is one of:

an atom	=	a template (input) file
T	=	the terminal

Side Effects:

SYM!-GENTRANIN processes each template file in *list-of-fnames* sequentially.

A template file may contain any number of parts, each of which is either an active or an inactive part. All active parts start with the character sequence **;BEGIN;** and end with **;END;**. The end of the template file is indicated by an extra **;END;** character sequence.

Inactive parts of template files are assumed to contain code in the target language (FORTRAN, RATFOR, PASCAL or C, depending on the value of the global varibale **GENTRANLANG!***).

All inactive parts are copied to the output. Comments delimited by the appropriate characters are also copied in their entirety to the output. Thus the character sequences **;BEGIN;** and **;END;** have no special meanings within comments.

Active parts may contain any number of REDUCE expressions, statements, and commands. They are not copied directly to the output. Instead, they are given to REDUCE for evaluation in algebraic mode[15]. All output generated by each evaluation is sent to the file(s) currently selected for output. Returned values are only printed on the terminal.

Active parts will most likely contain calls to GENTRAN to generate code. This means that the result of processing a template file will be the original template file with all active parts replaced by generated code.

Returned Value:

SYM!-GENTRANIN returns the name(s) of the file(s) to which code was written. If code was written to one file, the returned value is an atom; otherwise, it is a list.

Diagnostic Messages:

```
*** OUTPUT FILE ALREADY EXISTS

   OVERWRITE FILE? (Y/N)

***** NONEXISTENT INPUT FILE

***** TEMPLATE FILE ALREADY OPEN FOR INPUT

***** WRONG TYPE OF ARG
```

7.3 Output Redirection

Section 4 describes four slgebraic mode commands which select, open, and close output files. The algebraic mode commands **GENTRANOUT**, **GEN-**

[15] Active parts are evaluated in algebraic mode unless the mode is explicitly changed to symbolic from within the active part itself. This is true regardless of which mode the system was in when the template processor was called.

TRANSHUT, **GENTRANPUSH**, and **GENTRANPOP** are analogous to the symbolic mode **SYM!-GENTRANOUT**, **SYM!-GENTRANSHUT**, **SYM!-GENTRANPUSH**, and **SYM!-GENTRANPOP** functions, respectively.

7.3.1 SYM!-GENTRANOUT

Syntax:

SYM!-GENTRANOUT *list-of-fnames*;

Function Type:

expr

Argument:

list-of-fnames evaluates to a LISP list containing one or more *fname*s, where each *fname* is one of:

an atom	=	an output file
T	=	the terminal
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN

Side Effect:

GENTRAN maintains a list of files currently open for output by GENTRAN *only*. **SYM!-GENTRANOUT** inserts each file name represented in *list-of-fnames* into that list and opens each one for output. It also resets the currently selected output file(s) to be all of the files represented in *list-of-fnames*.

Returned Value:

SYM!-GENTRANOUT returns the name(s) of the file(s) represented by *list-of-fnames*; i.e., the current output file(s) after the command has been executed. If there is only one file selected for output, the returned value is an atom; otherwise, it is a list.

Diagnostic Messages:

```
*** OUTPUT FILE ALREADY EXISTS
```

```
   OVERWRITE FILE? (Y/N)

***** WRONG TYPE OF ARG
```

7.3.2 SYM!-GENTRANSHUT

Syntax:

SYM!-GENTRANSHUT *list-of-fnames* ;

Function Type:

expr

Argument:

list-of-fnames evaluates to a LISP list containing one or more *fnames*, where each *fname* is one of:

an atom	=	an output file
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN

Side Effects:

SYM!-GENTRANSHUT creates a list of file names from *list-of-fnames*, deletes each from the output file list, and closes the corresponding files. If (all of) the current output file(s) are closed, then the current output file is reset to the terminal.

Returned Value:

SYM!-GENTRANSHUT returns the name(s) of the file(s) selected for output after the command has been executed. If there is only one file selected for output, the returned value is an atom; otherwise, it is a list.

Diagnostic Messages:

```
*** FILE NOT OPEN FOR OUTPUT

***** WRONG TYPE OF ARG
```

7.3.3 SYM!-GENTRANPUSH

Syntax:

SYM!-GENTRANPUSH *list-of-fnames*;

Function Type:

expr

Argument:

list-of-fnames evaluates to a LISP list containing one or more *fname*s, each of which is one of:

an atom	=	an output file
T	=	the terminal
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN

Side Effects:

SYM!-GENTRANPUSH creates a list of file name(s) from *lis-of-fnames* and pushes that list onto the output stack. Each file in the list that is not already open for output is opened at this time. The current output file is reset to this new element on the top of the stack.

Returned Value:

SYM!-GENTRANPUSH returns the name(s) of the file(s) represented by *list-of-fnames*; i.e., the current output file(s) after the command has been executed. If there is only one file selected for output, the returned value is an atom; otherwise, it is a list.

Diagnostic Messages:

```
*** OUTPUT FILE ALREADY EXISTS

   OVERWRITE FILE? (Y/N)

***** WRONG TYPE OF ARG
```

7.3.4 SYM!-GENTRANPOP

Syntax:

SYM!-GENTRANPOP *list-of-fnames*;

Function Type:

expr

Argument:

list-of-fnames evaluates to a LISP list containing one or more *fname*s, where each *fname* is one of:

an atom	=	an output file
T	=	the terminal
NIL	=	the current output file(s)
ALL!*	=	all files currently open for output by GENTRAN

Side Effects:

SYM!-GENTRANPOP deletes the top-most occurrence of the single element containing the file name(s) represented by *list-of-fnames* from the output stack. Files whose names have been completely removed from the output stack are closed. The current output file is reset to the (new) element on the top of the output stack.

Returned Value:

SYM!-GENTRANPOP returns the name(s) of the file(s) selected for output after the command has been executed. If there is only one file selected for output, the returned value is an atom; otherwise, it is a list.

Diagnostic Messages:

```
*** FILE NOT OPEN FOR OUTPUT

***** WRONG TYPE OF ARG
```

8 Translatable REDUCE Expressions & Statements

A substantial subset of all REDUCE expressions and statements can be translated by GENTRAN into semantically equivalent code in the target numerical language[16]. This section contains examples and a formal definition of translatable REDUCE expressions and statements.

8.1 Examples of Translatable Statements

The following three tables contain listings of REDUCE statement types that can be translated by GENTRAN. An example of each statement type is shown, and FORTRAN, RATFOR, PASCAL and C code generated for each example is also shown.

8.2 Formal Definition

The remainder of this section contains a formal definition of all REDUCE expressions, statements, and prefix forms that can be translated by GENTRAN into FORTRAN, RATFOR, PASCAL and C code.

Preliminary Definitions

An *id* is an identifier. Certain *id*'s are reserved words and may not be used as array names or subprogram names. The complete list appears in the *Reserved Words* section.

A *string* consists of any number of characters (excluding double quotes) which are enclosed in double quotes.

Reserved Words

The following reserved words may not be used as array names or subprogram names[17]:

[16] It should be noted that call-by-value parameter passing is used in REDUCE, whereas call-by-address parameter passing is normally used in FORTRAN and RATFOR. GENTRAN does *not* attempt to simulate call-by-value passing in FORTRAN and RATFOR, although this could be done by generating temporary variables, assigning values to them, and using them in subprogram calls.

[17] Note that names of other built-in REDUCE functions *can* be translated, but remember that they will be translated *literally* unless **EVAL**'d first. For example: **GENTRAN DERIV := DF(2*X^2-X-1, X)$** generates `DERIV=DF(2*X**2-X-1,X)` whereas

TYPE	EXAMPLE	FORTRAN CODE
simple	**V:=X^2+X$**	`V=X**2+X`

matrix	**M:=MAT((U,V),
	(W,X))$**
		`M(1,1)=U`
		`M(1,2)=V`
		`M(2,1)=W`
		`M(2,2)=X`

sum	**S:=FOR I:=1:10
	SUM V(I)$**
		`S=0.0`
		`DO 25001 I=1,10`
		`  S=S+V(I)`
		`25001 CONTINUE`

product	**P:=FOR I:=2 STEP 2
	UNTIL N
	PRODUCT I$**
		`P=1`
		`DO 25002 I=2,N,2`
		`  P=P*I`
		`25002 CONTINUE`

conditional	**X := IF A<B THEN
	A ELSE B$**
		`IF (A.LT.B) THEN`
		`  X=A`
		`ELSE`
		`  X=B`
		`ENDIF`

Table 1: REDUCE assignments translatable to FORTRAN

TYPE	EXAMPLE	FORTRAN CODE
for	**FOR I:=1:8 DO** **V(I):=0.0$**	DO 25003 I=1,8 V(I)=0.0 25003 CONTINUE
while	**WHILE F(N)>0.0 DO** **N:=N+1$**	25004 IF(.NOT.F(N).GT.0.0) . GOTO 25005 N=N+1 GOTO 25004 25005 CONTINUE
repeat	**REPEAT X:=X/2.0** **UNTIL F(X)<0.0$**	25006 CONTINUE X=X/2.0 IF(.NOT.F(X).LT.0.0) . GOTO 25006

Table 2: REDUCE Loop structures translatable to FORTRAN

AND, BLOCK, COND, DIFFERENCE, EQUAL, EXPT, FOR, GEQ, GO, GREATERP, LEQ, LESSP, MAT, MINUS, NEQ, NOT, OR, PLUS, PROCEDURE, PROGN, QUOTIENT, RECIP, REPEAT, RETURN, SETQ, TIMES, WHILE, WRITE

8.2.1 Translatable REDUCE Expressions and Statements

Expressions

GENTRAN DERIV :=: DF(2*X^2-X-1, X)$ generates `DERIV=4*X-1`

TYPE	EXAMPLE	FORTRAN CODE
Conditionals:		

if	**IF X>0.0**
	THEN Y:=X$
		`IF (X.GT.0.0) THEN`
		`Y=X`
		`ENDIF`

if - else	**IF X>0.0 THEN Y:=X**
	ELSE Y:=-X$
		`IF (X.GT.0.0) THEN`
		`Y=X`
		`ELSE`
		`Y=-X`
		`ENDIF`

Unconditional Transfer of Control:		
goto	**GOTO LOOP$**	`GOTO 25010`
call	**CALCV(V,X,Y,Z)$**	`CALL CALCV(V,X,Y,Z)`
return	**RETURN X^2$**	*functionname*`=X**2` `RETURN`
Sequences & Groups:		
sequence	**<< U:=X^2;** **V:=Y^2>>$**	`U=X**2` `V=Y**2`
group	**BEGIN** **U:=X^2;** **V:=Y^2** **END$**	`U=X**2` `V=Y**2`

Table 3: REDUCE control structures translatable to FORTRAN

TYPE	EXAMPLE	RATFOR CODE
Assignments:		
simple	**V:=X^2+X$**	`V=X**2+X`

matrix	**M:=MAT((U,V),(W,X))$**	`M(1,1)=U`
		`M(1,2)=V`
		`M(2,1)=W`
		`M(2,2)=X`

sum	**S:=FOR I:=1:10**
	SUM V(I)$
		`S=0.0`
		`DO I=1,10`
		`  S=S+V(I)`

product	**P:=FOR I:=2 STEP 2**
	UNTIL N
	PRODUCT I$
		`P=1`
		`DO I=2,N,2`
		`  P=P*I`

conditional	**X := IF A<B THEN**
	A ELSE B$
		`IF (A<B)`
		`  X=A`
		`ELSE`
		`  X=B`

Control Structures:		
Loops:		
for	**FOR I:=1:8 DO** **V(I):=0.0$**	`DO I=1,8` `  V(I)=0.0`
while	**WHILE F(N)>0.0 DO** **N:=N+1$**	`WHILE(F(N)>0.0)` `  N=N+1`

repeat	**REPEAT X:=X/2.0**
	UNTIL F(X)<0.0$
		`REPEAT`
		`  X=X/2.0`
		`UNTIL(F(X)<0.0)`

Table 4: REDUCE forms translatable to RATFOR

TYPE	EXAMPLE	RATFOR CODE
Conditionals:		
if	**IF X>0.0 THEN Y:=X$**	`IF(X>0.0)` `  Y=X`

if - else	**IF X>0.0 THEN Y:=X**
	ELSE Y:=-X$
		`IF(X>0.0)`
		`  Y=X`
		`ELSE`
		`  Y=-X`

Unconditional Transfer of Control:		
goto	**GOTO LOOP$**	`GOTO 25010`
call	**CALCV(V,X,Y,Z)$**	`CALL CALCV(V,X,Y,Z)`
return	**RETURN X^2$**	`RETURN(X**2)`
Sequences & Groups:		
sequence	**<< U:=X^2;V:=Y^2>>$**	`U=X**2` `V=Y**2`

group	**BEGIN**
	U:=X^2;
	V:=Y^2
	END$
		`{`
		`  U=X**2`
		`  V=Y**2`
		`}`

Table 5: REDUCE forms translatable to RATFOR

TYPE	EXAMPLE	PASCAL CODE
Assignments:		
simple	**V:=X^2+X$**	`V=X**2+X;`

matrix	**M:=MAT((U,V),
	(W,X))$**
		`BEGIN`
		`  M(1,1)=U;`
		`  M(1,2)=V;`
		`  M(2,1)=W;`
		`  M(2,2)=X;`
		`END;`

sum	**S:=FOR I:=1:10
	SUM V(I)$**
		`BEGIN`
		`  S=0.0`
		`  FOR I:=1 TO 10 DO`
		`  S:=S+V(I)`
		`END;`

product	**P:=FOR I:=2:N
	PRODUCT I$**
		`BEGIN`
		`  P:=1;`
		`  FOR I:=2 TO N DO`
		`  P:=P*I`
		`END;`

conditional	**X := IF A<B THEN
	A ELSE B$**
		`IF (A<B) THEN`
		`  X:=A;`
		`ELSE`
		`  X:=B;`

Table 6: REDUCE forms translatable to PASCAL

TYPE	EXAMPLE	PASCAL CODE
Control Structures:		
Loops:		
for	**FOR I:=1:8 DO** **V(I):=0.0$**	`FOR I:=1 TO 8 DO` `V(I):=0.0;`
while	**WHILE F(N)>0.0 DO** **N:=N+1$**	`WHILE (F(N)>0.0)` `N:=N+1.0;`

repeat	**REPEAT X:=X/2.0**
	UNTIL F(X)<0.0$
		`REPEAT`
		`X:=X/2.0`
		`UNTIL F(X)<0.0;`

Table 7: REDUCE forms translatable to PASCAL

Arithmetic Expressions:

exp ::= *number* | var | funcall | - exp | / exp | exp + exp | exp - exp | exp * exp | exp / exp | exp ** exp | exp ^ exp | (exp)

var ::= *id* | *id* (exp_1, exp_2, ... , exp_n) $n > 0$

funcall ::= *id* (arg_1, arg_2, ... , arg_n) $n \geq 0$

arg ::= exp | logexp | *string*

Logical Expressions:

logexp ::= *T* | *NIL* | var | funcall | exp > exp | exp >= exp | exp = exp | exp *NEQ* exp | exp < exp | exp <= exp | *NOT* logexp | logexp *AND* logexp | logexp *OR* logexp | (logexp)

TYPE	EXAMPLE	PASCAL CODE
Conditionals:		
if	**IF X>0.0 THEN Y:=X$**	`IF X>0.0 THEN` `  Y:=X;`

if - else	**IF X>0.0 THEN Y:=X**
	ELSE Y:=-X$
		`IF X>0.0 THEN`
		`  Y:=X;`
		`ELSE`
		`  Y:=-X;`

Unconditional Transfer of Control:		
goto	**GOTO LOOP$**	`GOTO 25010;`
call	**CALCV(V,X,Y,Z)$**	`CALCV(V,X,Y,Z);`

return	**RETURN X^2$**	*functionname*`=X**2;`
		`GOTO 99999{RETURN}`
		`99999;`

Sequences & Groups:		

sequence	**<< U:=X^2;V:=Y^2>>$**	`BEGIN`
		`  U:=X**2;`
		`  V:=Y**2`
		`END;`

group	**BEGIN**
	U:=X^2;
	V:=Y^2
	END$
		`BEGIN`
		`  U:=X**2;`
		`  V:=Y**2`
		`END`

Table 8: REDUCE forms translatable to PASCAL

TYPE	EXAMPLE	C CODE
Assignments:		
simple	**V:=X^2+X$**	`V=power(X,2)+X;`

matrix	**M:=MAT((U,V),(W,X))$**	`M[1][1]=U;`
		`M[1][2]=V;`
		`M[2][1]=W;`
		`M[2][2]=X;`

sum	**S:=FOR I:=1:10**
	SUM V(I)$
		`S=0.0;`
		`for(I=1;I<=10;++I)`
		`  S+=V[I];`

product	**P:=FOR I:=2 STEP 2**
	UNTIL N
	PRODUCT I$
		`P=1;`
		`for(I=2;I<=N;++I)`
		`  P*=I;`

conditional	**X := IF A<B THEN**
	A ELSE B$
		`if (A<B)`
		`  X=A;`
		`else`
		`  X=B;`

Control Structures:		
Loops:		
for	**FOR I:=1:8 DO** **V(I):=0.0$**	`for(I=1;I<=8;++I)` `  V[I]=0.0;`
while	**WHILE F(N)>0.0 DO** **N:=N+1$**	`while(F(N)>0.0)` `  N+=1;`

repeat	**REPEAT X:=X/2.0**
	UNTIL F(X)<0.0$
		`do`
		`  X/=2.0;`
		`while(F(X)>=0.0);`

Table 9: REDUCE forms translatable to C

TYPE	EXAMPLE	C CODE
Conditionals:		
if	**IF X>0.0 THEN Y:=X$**	`if(X>0.0)` `  Y=X;`

if - else	**IF X>0.0 THEN Y:=X**
	ELSE Y:=-X$
		`if(X>0.0)`
		`  Y=X;`
		`else`
		`  Y=-X;`

Unconditional Transfer of Control:		
goto	**GOTO LOOP$**	`goto LOOP;`
call	**CALCV(V,X,Y,Z)$**	`CALCV(V,X,Y,Z);`
return	**RETURN X^2$**	`return(power(X,2) );`
Sequences & Groups:		
sequence	<< **U:=X^2;V:=Y^2**>>$	`U=power(X,2);` `V=power(Y,2);`

group	**BEGIN**
	U:=X^2;
	V:=Y^2
	END$
		`{`
		`  U=power(x,2);`
		`  V=power(Y,2);`
		`}`

Table 10: REDUCE forms translatable to C

Operator Precedence

The following is a list of REDUCE arithmetic and logical operators in order of decreasing precedence:

** (or ^) / * — + < <= > >= NEQ = NOT AND OR

When unparenthesised expressions are translated which contain operators whose precedence in REDUCE differs from that in the target language, parentheses are automatically generated. Thus the meaning of the original expression is preserved[18].

Statements

stmt ::= assign | break | cond | while | repeat | for | goto | label | call | return | stop | stmtgp

Assignment Statements:

assign ::= var := assign' | matassign | cond

assign' ::= exp | logexp

matassign ::= $id := MAT($ $(\exp_{11}, \ldots, \exp_{1m})$, $(\exp_{21}, \ldots, \exp_{2m})$, : : $(\exp_{n1}, \ldots, \exp_{nm}))$ $n, m > 0$

Break Statement:

break ::= *BREAK()*

Conditional Statements:

cond ::= *IF* logexp *THEN* stmt
IF logexp *THEN* stmt *ELSE* stmt

Loops:

[18] For example in REDUCE, **NOT A = B** and **NOT (A = B)** are equivalent, whereas in C, **! A == B** and **(!A) == B** are equivalent. Therefore, **NOT A = B** is translated into C code which forces the REDUCE precedence rules: **!(A == B)**

while ::= *WHILE* logexp *DO* stmt

repeat ::= *REPEAT* stmt *UNTIL* logexp

for ::= *FOR* var := exp *STEP* exp *UNTIL* exp *DO* stmt |
FOR var := exp *UNTIL* exp *DO* stmt |
FOR var := exp : exp *DO* stmt |
var := for' |

for' ::= var := for' |
FOR var := exp *STEP* exp *UNTIL* exp *SUM* exp |
FOR var := exp *UNTIL* exp *SUM* exp |
FOR var := exp : exp *SUM* exp |
FOR var := exp *STEP* exp *UNTIL* exp
PRODUCT exp |
FOR var := exp *UNTIL* exp *PRODUCT* exp |
FOR var := exp : exp *PRODUCT* exp

Goto Statement:

goto ::= *GOTO* label | *GO TO* label
label ::= *id* :

Subprogram Calls & Returns [19]:

call ::= *id* (arg_1, arg_2, ... , arg_n) $n \geq 0$

return ::= *RETURN* | *RETURN* arg

Stop & Exit Statements [20]:

stop ::= *STOP*()

Statement Groups [21]:

[19] Note that return statements can only be translated from inside of procedure definitions. The LITERAL function must be used to generate a return statement from anywhere else.

[20] In certain cases it may be convenient to generate a FORTRAN STOP statement or a C EXIT statement. Since there is no semantically equivalent REDUCE statement, STOP() can be used and will be translated appropriately.

[21] Note that REDUCE BEGIN... END statement groups are translated into RATFOR or C {... } statement groups, whereas REDUCE <<... >> statement groups are translated into RATFOR or C statement *sequences*. When the target language is FORTRAN, both types of REDUCE statement groups are translated into statement sequences.

stmtgp ::= << stmt_1 ; stmt_2 ; ... ; stmt_n >> |
BEGIN stmt_1 ; stmt_2 ; ... ; stmt_n *END* $n > 0$

Subprogram Definitions

defn ::= *PROCEDURE id* (id_1, id_2, ... , id_n) ; stmt |
PROCEDURE id (id_1, id_2, ... , id_n) ; exp $n \geq 0$

8.2.2 Translatable REDUCE Prefix Forms

Expressions

Arithmetic Expressions:

exp ::= *number* | funcall | var | (*DIFFERENCE* exp exp) |
(*EXPT* exp exp) | (*MINUS* exp) | (*PLUS* exp exp') |
(*QUOTIENT* exp exp) | (*RECIP* exp) |
(*TIMES* exp exp exp') | (*!*SQ* sqform)

where sqform is a standard quotient form equivalent to any acceptable prefix form.

exp' ::= exp_1 exp_2 ... exp_n $n \geq 0$

Logical Expressions:

logexp ::= *NIL* | *T* | funcall | var |
(*AND* logexp logexp logexp') | (*EQUAL* exp exp) |
(*GEQ* exp exp) | (*GREATERP* exp exp) |
(*LEQ* exp exp) | (*LESSP* exp exp) |
(*NEQ* exp exp) | (*NOT* logexp) |
(*OR* logexp logexp logexp')

logexp' ::= logexp_1 logexp_2 ... logexp_n $n \geq 0$

Statements

stmt ::= assign | break | call | cond | for | goto |
label | read | repeat | return | stmtgp |
stop | while | write

stmt' ::= stmt_1 stmt_2 ... stmt_n $n \geq 0$

Assignment Statements:

assign ::= (*SETQ* var exp) | (*SETQ* var logexp) | (*SETQ* id
(*MAT* list list'))

Conditional Statements:

cond ::= (*COND* (logexp stmt) cond1)

cond1 ::= (logexp stmt_1) ... (logexp stmt_n) $n \geq 0$

Loops:

for ::= (*FOR* var (exp exp exp) *DO* stmt) |
(*SETQ* var (*FOR* var (exp exp exp) *SUM* exp) |
(*SETQ* var (*FOR* var (exp exp exp)
PRODUCT exp)

repeat ::= (*REPEAT* stmt logexp)

while ::= (*WHILE* logexp stmt)

Go To Statements:

break ::= (*BREAK*)

goto ::= (*GO* label)

label ::= *id*

Subprogram Calls & Returns:

call ::= (*id* arg')

return ::= (*RETURN*) | (*RETURN* arg)

Stop & Exit Statements:

stop ::= (*STOP*)

Statement Groups:

stmtgp ::= (*PROGN* stmt stmt') | (*BLOCK* (id') stmt')

I/O Statements:

read ::= (*SETQ* var (*READ*))

write ::= (*WRITE* arg arg')

Subprogram Definitions:

defn ::= (*PROCEDURE id NIL EXPR* (id') stmt)

Miscellaneous

funcall	::=	(*id* arg')
var	::=	*id* | (*id* exp exp')
arg	::=	*string* | exp | logexp
arg'	::=	arg_1 arg_2 ... arg_n $n \geq 0$
list	::=	(exp exp')
list'	::=	list_1 list_2 ... list_n $n \geq 0$
id'	::=	id_1 id_2 ... id_n $n \geq 0$

9 List of Commands, Switches, & Variables

COMMANDS

GENTRAN *stmt* [**OUT***f1,f2,... ,fn*];
GENTRANIN *f1,f2,... ,fm* [**OUT***f1,f2,... ,fn*];
GENTRANOUT *f1,f2,... ,fn;*
GENTRANSHUT *f1,f2,... ,fn;*
GENTRANPUSH *f1,f2,... ,fn;*
GENTRANPOP *f1,f2,... ,fn;*

SPECIAL FUNCTIONS & OPERATORS

EVAL *exp*
var **::=** *exp;*
var **:=:** *exp;*
var **::=:** *exp;*
var **LSETQ** *exp;*
var **RSETQ** *exp;*
var **LRSETQ** *exp;*
DECLARE *v1,v2,... ,vn*: *type;*

DECLARE
<<
v11,v12,... ,v1n **:** *type1***;**
v12,v22,... ,v2n **:** *type2***;**
:
:
vm1,vm2,... ,vmn **:** *typen***;**
>>;

LITERAL *arg1,arg2,... ,argn;*

MODE SWITCHES

PERIOD

GENTRANSEG

GENDECS

DOUBLE

MAKECALLS

KEEPDECS

GETDECS

VARIABLES

GENTRANLANG!*

MAXEXPPRINTLEN!*

TEMPVARNAME!*

TEMPVARNUM!*

TEMPVARTYPE!*

GENSTMTNUM!*

GENSTMTINCR!*

TABLEN!*

FORTLINELEN!*

RATLINELEN!*

CLINELEN!*

PASCLINELEN!*

MINFORTLINELEN!*

MINRATLINELEN!*

MINCLINELEN!*

MINPASCLINELEN!*

DEFTYPE!*

TEMPORARY VARIABLE GENERATION, MARKING & UNMARKING

TEMPVAR *type;*

MARKVAR *var;*

UNMARKVAR *var;*

EXPLICIT GENERATION OF TYPE DECLARATIONS

GENDECS *subprogname;*

SYMBOLIC MODE FUNCTIONS

SYM!-GENTRAN *form;*

SYM!-GENTRANIN *list-of-fnames;*

SYM!-GENTRANOUT *list-of-fnames;*

SYM!-GENTRANSHUT *list-of-fnames;*

SYM!-GENTRANPUSH *list-of-fnames;*

SYM!-GENTRANPOP *list-of-fnames;*

SYMBOLIC MODE SPECIAL FORMS

(**DECLARE** (*type1 v11 v12 ... v1n*)
(*type2 v21 v22 ... v2n*)
:
:
(*typen vn1 vn2 ... vnn*))

(**LITERAL** *arg1 arg2 ... argn*)

(**EVAL** *exp*)

(**LSETQ** *var exp*)

(**RSETQ** *var exp*)

(**LRSETQ** *var exp*)

10 The Programs M1.F and M2.F.

This section contains the two files generated in chapter 6. Contents of file m1.f:

```
      M(1,1)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(DSIN(DBLE(
     . Q3))**2*Y*J30)+DSIN(DBLE(Q3))**2*J30Z+18.0D0*DCOS(DBLE
     . (Q3))*DCOS(DBLE(Q2))*P**2*M30+18.0D0*P**2*M30+P**2*M10
     . +J30Y+J10Y
      M(1,2)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(DSIN(DBLE(
     . Q3))**2*J30Y)+DSIN(DBLE(Q3))**2*J30Z+9.0D0*DCOS(DBLE(
     . Q3))*DCOS(DBLE(Q2))*P**2*M30+9.0D0*P**2*M30+J30Y
      M(1,3)=-(9.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*P**2*M30)
      M(2,2)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(DSIN(DBLE(
     . Q3))**2*J30Y)+DSIN(DBLE(Q3))**2*J30Z+9.0D0*P**2*M30+
     . J30Y
      M(2,3)=0.0D0
      M(3,3)=9.0D0*P**2*M30+J30X
      MIV(1,1)=(-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2)-(
     . 9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y)+9.0D0*DSIN(DBLE
     . (Q3))**2*P**2*M30*J30Z-(9.0D0*DSIN(DBLE(Q3))**2*P**2*
     . M30*J30X)-(DSIN(DBLE(Q3))**2*J30Y*J30X)+DSIN(DBLE(Q3))
     . **2*J30Z*J30X+81.0D0*P**4*M30**2+9.0D0*P**2*M30*J30Y+
     . 9.0D0*P**2*M30*J30X+J30Y*J30X)/(729.0D0*DSIN(DBLE(Q3))
     . **4*DSIN(DBLE(Q2))**2*P**6*M30**3+81.0D0*DSIN(DBLE(Q3
     . ))**4*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Y-(81.0D0*DSIN(
     . DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Z)+
     . 81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*M30**2*J30-(81.0D0*
     . DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)+9.0D0*DSIN(DBLE(Q3
     . ))**4*P**2*Y*M30*J30Y*J30-(9.0D0*DSIN(DBLE(Q3))**4*P**
     . 2*Y*M30*J30Z*J30)+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*
     . J30X*J30-(9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN
     . (DBLE(Q3))**4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*
     . J30Y*J30X*J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(
     . DSIN(DBLE(Q3))**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y
     . *J30Z*J30X-(729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**
     . 2*P**6*M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2)
     . )**2*P**4*M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6
```

```
     . *M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*
     . DSIN(DBLE(Q3))**2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P
     . **4*M30**2*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*
     . J30Y*M10)+9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(
     . 9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN
     . (DBLE(Q3))**2*P**2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3
     . ))**2*P**2*Y*M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**
     . 2*M30*J30Y**2-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*
     . J10Y)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y+9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0*DSIN(DBLE
     . (Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))**2*P**2*
     . J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*M10*J30X-(
     . DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE(Q3))**2*
     . J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*J30X)+DSIN(
     . DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(DBLE(Q3))**2
     . *DCOS(DBLE(Q2))**2*P**6*M30**3)-(81.0D0*DCOS(DBLE(Q3))
     . **2*DCOS(DBLE(Q2))**2*P**4*M30**2*J30X)+729.0D0*P**6*
     . M30**3+81.0D0*P**6*M30**2*M10+81.0D0*P**4*M30**2*J30Y+
     . 81.0D0*P**4*M30**2*J10Y+81.0D0*P**4*M30**2*J30X+9.0D0*
     . P**4*M30*J30Y*M10+9.0D0*P**4*M30*M10*J30X+9.0D0*P**2*
     . M30*J30Y*J10Y+9.0D0*P**2*M30*J30Y*J30X+9.0D0*P**2*M30*
     . J10Y*J30X+P**2*J30Y*M10*J30X+J30Y*J10Y*J30X)
      MIV(1,2)=(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2+9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*M30*J30Y-(9.0D0*DSIN(DBLE(Q3))
     . **2*P**2*M30*J30Z)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*
     . J30X+DSIN(DBLE(Q3))**2*J30Y*J30X-(DSIN(DBLE(Q3))**2*
     . J30Z*J30X)-(81.0D0*DCOS(DBLE(Q3))*DCOS(DBLE(Q2))*P**4*
     . M30**2)-(9.0D0*DCOS(DBLE(Q3))*DCOS(DBLE(Q2))*P**2*M30*
     . J30X)-(81.0D0*P**4*M30**2)-(9.0D0*P**2*M30*J30Y)-(
     . 9.0D0*P**2*M30*J30X)-(J30Y*J30X))/(729.0D0*DSIN(DBLE(
     . Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30**3+81.0D0*DSIN(DBLE
     . (Q3))**4*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Y-(81.0D0*
     . DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Z)+
     . 81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*M30**2*J30-(81.0D0*
     . DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)+9.0D0*DSIN(DBLE(Q3
     . ))**4*P**2*Y*M30*J30Y*J30-(9.0D0*DSIN(DBLE(Q3))**4*P**
```

```
     . 2*Y*M30*J30Z*J30)+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*
     . J30X*J30-(9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN
     . (DBLE(Q3))**4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*
     . J30Y*J30X*J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(
     . DSIN(DBLE(Q3))**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y
     . *J30Z*J30X-(729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**
     . 2*P**6*M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2)
     . )**2*P**4*M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6
     . *M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*
     . DSIN(DBLE(Q3))**2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P
     . **4*M30**2*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*
     . J30Y*M10)+9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(
     . 9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN
     . (DBLE(Q3))**2*P**2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3
     . ))**2*P**2*Y*M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**
     . 2*M30*J30Y**2-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*
     . J10Y)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y+9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0*DSIN(DBLE
     . (Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))**2*P**2*
     . J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*M10*J30X-(
     . DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE(Q3))**2*
     . J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*J30X)+DSIN(
     . DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(DBLE(Q3))**2
     . *DCOS(DBLE(Q2))**2*P**6*M30**3)-(81.0D0*DCOS(DBLE(Q3))
     . **2*DCOS(DBLE(Q2))**2*P**4*M30**2*J30X)+729.0D0*P**6*
     . M30**3+81.0D0*P**6*M30**2*M10+81.0D0*P**4*M30**2*J30Y+
     . 81.0D0*P**4*M30**2*J10Y+81.0D0*P**4*M30**2*J30X+9.0D0*
     . P**4*M30*J30Y*M10+9.0D0*P**4*M30*M10*J30X+9.0D0*P**2*
     . M30*J30Y*J10Y+9.0D0*P**2*M30*J30Y*J30X+9.0D0*P**2*M30*
     . J10Y*J30X+P**2*J30Y*M10*J30X+J30Y*J10Y*J30X)
      MIV(1,3)=(-(81.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**
     . 4*M30**2)-(9.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**2
     . *M30*J30Y)+9.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**2
     . *M30*J30Z+81.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*P**4*
     . M30**2+9.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*P**2*M30*
```

```
     . J30Y)/(729.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**
     . 6*M30**3+81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P
     . **4*M30**2*J30Y-(81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2
     . ))**2*P**4*M30**2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*
     . Y*M30**2*J30-(81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*
     . J30Y)+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Z*J30)+9.0D0*
     . DSIN(DBLE(Q3))**4*P**2*Y*M30*J30X*J30-(9.0D0*DSIN(DBLE
     . (Q3))**4*P**2*M30*J30Y**2)+9.0D0*DSIN(DBLE(Q3))**4*P**
     . 2*M30*J30Y*J30Z-(9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y
     . *J30X)+DSIN(DBLE(Q3))**4*Y*J30Y*J30X*J30-(DSIN(DBLE(Q3
     . ))**4*Y*J30Z*J30X*J30)-(DSIN(DBLE(Q3))**4*J30Y**2*J30X
     . )+DSIN(DBLE(Q3))**4*J30Y*J30Z*J30X-(729.0D0*DSIN(DBLE(
     . Q3))**2*DSIN(DBLE(Q2))**2*P**6*M30**3)-(81.0D0*DSIN(
     . DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Y)-(
     . 729.0D0*DSIN(DBLE(Q3))**2*P**6*M30**3)-(81.0D0*DSIN(
     . DBLE(Q3))**2*P**6*M30**2*M10)-(81.0D0*DSIN(DBLE(Q3))**
     . 2*P**4*Y*M30**2*J30)+81.0D0*DSIN(DBLE(Q3))**2*P**4*M30
     . **2*J30Z-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*J10Y)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*J30X)-(9.0D0*DSIN
     . (DBLE(Q3))**2*P**4*M30*J30Y*M10)+9.0D0*DSIN(DBLE(Q3))
     . **2*P**4*M30*J30Z*M10-(9.0D0*DSIN(DBLE(Q3))**2*P**4*
     . M30*M10*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*M30*J30Y
     . *J30)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*M30*J30X*J30)+
     . 9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y**2-(9.0D0*DSIN(
     . DBLE(Q3))**2*P**2*M30*J30Y*J10Y)+9.0D0*DSIN(DBLE(Q3))
     . **2*P**2*M30*J30Z*J10Y+9.0D0*DSIN(DBLE(Q3))**2*P**2*
     . M30*J30Z*J30X-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J10Y*
     . J30X)-(DSIN(DBLE(Q3))**2*P**2*J30Y*M10*J30X)+DSIN(DBLE
     . (Q3))**2*P**2*J30Z*M10*J30X-(DSIN(DBLE(Q3))**2*Y*J30Y*
     . J30X*J30)+DSIN(DBLE(Q3))**2*J30Y**2*J30X-(DSIN(DBLE(Q3
     . ))**2*J30Y*J10Y*J30X)+DSIN(DBLE(Q3))**2*J30Z*J10Y*J30X
     . -(729.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**6*M30
     . **3)-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*
     . M30**2*J30X)+729.0D0*P**6*M30**3+81.0D0*P**6*M30**2*
     . M10+81.0D0*P**4*M30**2*J30Y+81.0D0*P**4*M30**2*J10Y+
     . 81.0D0*P**4*M30**2*J30X+9.0D0*P**4*M30*J30Y*M10+9.0D0*
     . P**4*M30*M10*J30X+9.0D0*P**2*M30*J30Y*J10Y+9.0D0*P**2*
```

```
     . M30*J30Y*J30X+9.0D0*P**2*M30*J10Y*J30X+P**2*J30Y*M10*
     . J30X+J30Y*J10Y*J30X)
      MIV(2,2)=(-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*
     . P**4*M30**2)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2)-(
     . 9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*M30*J30)+9.0D0*DSIN(
     . DBLE(Q3))**2*P**2*M30*J30Z-(9.0D0*DSIN(DBLE(Q3))**2*P
     . **2*M30*J30X)-(DSIN(DBLE(Q3))**2*Y*J30X*J30)+DSIN(DBLE
     . (Q3))**2*J30Z*J30X+162.0D0*DCOS(DBLE(Q3))*DCOS(DBLE(Q2
     . ))*P**4*M30**2+18.0D0*DCOS(DBLE(Q3))*DCOS(DBLE(Q2))*P
     . **2*M30*J30X+162.0D0*P**4*M30**2+9.0D0*P**4*M30*M10+
     . 9.0D0*P**2*M30*J30Y+9.0D0*P**2*M30*J10Y+18.0D0*P**2*
     . M30*J30X+P**2*M10*J30X+J30Y*J30X+J10Y*J30X)/(729.0D0*
     . DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30**3+81.0D0
     . *DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Y-
     . (81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*M30**
     . 2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*M30**2*J30-(
     . 81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)+9.0D0*DSIN(
     . DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(9.0D0*DSIN(DBLE(Q3))
     . **4*P**2*Y*M30*J30Z*J30)+9.0D0*DSIN(DBLE(Q3))**4*P**2*
     . Y*M30*J30X*J30-(9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y
     . **2)+9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0
     . *DSIN(DBLE(Q3))**4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))
     . **4*Y*J30Y*J30X*J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30
     . )-(DSIN(DBLE(Q3))**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*
     . J30Y*J30Z*J30X-(729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2
     . ))**2*P**6*M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE
     . (Q2))**2*P**4*M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*
     . P**6*M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10
     . )-(81.0D0*DSIN(DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*
     . DSIN(DBLE(Q3))**2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P
     . **4*M30**2*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*
     . J30Y*M10)+9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(
     . 9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN
     . (DBLE(Q3))**2*P**2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3
     . ))**2*P**2*Y*M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**
     . 2*M30*J30Y**2-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*
     . J10Y)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y+9.0D0
```

```
     . *DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0*DSIN(DBLE
     . (Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))**2*P**2*
     . J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*M10*J30X-(
     . DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE(Q3))**2*
     . J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*J30X)+DSIN(
     . DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(DBLE(Q3))**2
     . *DCOS(DBLE(Q2))**2*P**6*M30**3)-(81.0D0*DCOS(DBLE(Q3))
     . **2*DCOS(DBLE(Q2))**2*P**4*M30**2*J30X)+729.0D0*P**6*
     . M30**3+81.0D0*P**6*M30**2*M10+81.0D0*P**4*M30**2*J30Y+
     . 81.0D0*P**4*M30**2*J10Y+81.0D0*P**4*M30**2*J30X+9.0D0*
     . P**4*M30*J30Y*M10+9.0D0*P**4*M30*M10*J30X+9.0D0*P**2*
     . M30*J30Y*J10Y+9.0D0*P**2*M30*J30Y*J30X+9.0D0*P**2*M30*
     . J10Y*J30X+P**2*J30Y*M10*J30X+J30Y*J10Y*J30X)
      MIV(2,3)=(81.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**4*
     . M30**2+9.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**2*M30
     . *J30Y-(9.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**2*M30
     . *J30Z)-(81.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*DCOS(DBLE
     . (Q3))*DCOS(DBLE(Q2))*P**4*M30**2)-(81.0D0*DSIN(DBLE(Q3
     . ))*DSIN(DBLE(Q2))*P**4*M30**2)-(9.0D0*DSIN(DBLE(Q3))*
     . DSIN(DBLE(Q2))*P**2*M30*J30Y))/(729.0D0*DSIN(DBLE(Q3))
     . **4*DSIN(DBLE(Q2))**2*P**6*M30**3+81.0D0*DSIN(DBLE(Q3)
     . )**4*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Y-(81.0D0*DSIN(
     . DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Z)+
     . 81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*M30**2*J30-(81.0D0*
     . DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)+9.0D0*DSIN(DBLE(Q3
     . ))**4*P**2*Y*M30*J30Y*J30-(9.0D0*DSIN(DBLE(Q3))**4*P**
     . 2*Y*M30*J30Z*J30)+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*
     . J30X*J30-(9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN
     . (DBLE(Q3))**4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*
     . J30Y*J30X*J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(
     . DSIN(DBLE(Q3))**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y
     . *J30Z*J30X-(729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**
     . 2*P**6*M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2)
     . )**2*P**4*M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6
     . *M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*
     . DSIN(DBLE(Q3))**2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(
```

```
     . Q3))**2*P**4*M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P
     . **4*M30**2*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*
     . J30Y*M10)+9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(
     . 9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN
     . (DBLE(Q3))**2*P**2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3
     . ))**2*P**2*Y*M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**
     . 2*M30*J30Y**2-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*
     . J10Y)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y+9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0*DSIN(DBLE
     . (Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))**2*P**2*
     . J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*M10*J30X-(
     . DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE(Q3))**2*
     . J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*J30X)+DSIN(
     . DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(DBLE(Q3))**2
     . *DCOS(DBLE(Q2))**2*P**6*M30**3)-(81.0D0*DCOS(DBLE(Q3))
     . **2*DCOS(DBLE(Q2))**2*P**4*M30**2*J30X)+729.0D0*P**6*
     . M30**3+81.0D0*P**6*M30**2*M10+81.0D0*P**4*M30**2*J30Y+
     . 81.0D0*P**4*M30**2*J10Y+81.0D0*P**4*M30**2*J30X+9.0D0*
     . P**4*M30*J30Y*M10+9.0D0*P**4*M30*M10*J30X+9.0D0*P**2*
     . M30*J30Y*J10Y+9.0D0*P**2*M30*J30Y*J30X+9.0D0*P**2*M30*
     . J10Y*J30X+P**2*J30Y*M10*J30X+J30Y*J10Y*J30X)
      MIV(3,3)=(9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30-(9.0D0
     . *DSIN(DBLE(Q3))**4*P**2*M30*J30Y)+DSIN(DBLE(Q3))**4*Y*
     . J30Y*J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30)-(DSIN(DBLE(Q3)
     . )**4*J30Y**2)+DSIN(DBLE(Q3))**4*J30Y*J30Z-(81.0D0*DSIN
     . (DBLE(Q3))**2*P**4*M30**2)-(9.0D0*DSIN(DBLE(Q3))**2*P
     . **4*M30*M10)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*M30*J30)+
     . 9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z-(9.0D0*DSIN(DBLE
     . (Q3))**2*P**2*M30*J10Y)-(DSIN(DBLE(Q3))**2*P**2*J30Y*
     . M10)+DSIN(DBLE(Q3))**2*P**2*J30Z*M10-(DSIN(DBLE(Q3))**
     . 2*Y*J30Y*J30)+DSIN(DBLE(Q3))**2*J30Y**2-(DSIN(DBLE(Q3)
     . )**2*J30Y*J10Y)+DSIN(DBLE(Q3))**2*J30Z*J10Y-(81.0D0*
     . DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*M30**2)+
     . 81.0D0*P**4*M30**2+9.0D0*P**4*M30*M10+9.0D0*P**2*M30*
     . J30Y+9.0D0*P**2*M30*J10Y+P**2*J30Y*M10+J30Y*J10Y)/(
     . 729.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30**
     . 3+81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*M30
     . **2*J30Y-(81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P
```

```
     . **4*M30**2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*M30**
     . 2*J30-(81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)+
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(9.0D0*
     . DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Z*J30)+9.0D0*DSIN(DBLE
     . (Q3))**4*P**2*Y*M30*J30X*J30-(9.0D0*DSIN(DBLE(Q3))**4*
     . P**2*M30*J30Y**2)+9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*
     . J30Y*J30Z-(9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y*J30X)
     . +DSIN(DBLE(Q3))**4*Y*J30Y*J30X*J30-(DSIN(DBLE(Q3))**4*
     . Y*J30Z*J30X*J30)-(DSIN(DBLE(Q3))**4*J30Y**2*J30X)+DSIN
     . (DBLE(Q3))**4*J30Y*J30Z*J30X-(729.0D0*DSIN(DBLE(Q3))**
     . 2*DSIN(DBLE(Q2))**2*P**6*M30**3)-(81.0D0*DSIN(DBLE(Q3)
     . )**2*DSIN(DBLE(Q2))**2*P**4*M30**2*J30Y)-(729.0D0*DSIN
     . (DBLE(Q3))**2*P**6*M30**3)-(81.0D0*DSIN(DBLE(Q3))**2*P
     . **6*M30**2*M10)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*Y*M30**
     . 2*J30)+81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*J30Z-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*J10Y)-(81.0D0*
     . DSIN(DBLE(Q3))**2*P**4*M30**2*J30X)-(9.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30*J30Y*M10)+9.0D0*DSIN(DBLE(Q3))**2*P**
     . 4*M30*J30Z*M10-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*M10*
     . J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*M30*J30Y*J30)-(
     . 9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*M30*J30X*J30)+9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*M30*J30Y**2-(9.0D0*DSIN(DBLE(Q3
     . ))**2*P**2*M30*J30Y*J10Y)+9.0D0*DSIN(DBLE(Q3))**2*P**2
     . *M30*J30Z*J10Y+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*
     . J30X-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J10Y*J30X)-(
     . DSIN(DBLE(Q3))**2*P**2*J30Y*M10*J30X)+DSIN(DBLE(Q3))**
     . 2*P**2*J30Z*M10*J30X-(DSIN(DBLE(Q3))**2*Y*J30Y*J30X*
     . J30)+DSIN(DBLE(Q3))**2*J30Y**2*J30X-(DSIN(DBLE(Q3))**2
     . *J30Y*J10Y*J30X)+DSIN(DBLE(Q3))**2*J30Z*J10Y*J30X-(
     . 729.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**6*M30**
     . 3)-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*
     . M30**2*J30X)+729.0D0*P**6*M30**3+81.0D0*P**6*M30**2*
     . M10+81.0D0*P**4*M30**2*J30Y+81.0D0*P**4*M30**2*J10Y+
     . 81.0D0*P**4*M30**2*J30X+9.0D0*P**4*M30*J30Y*M10+9.0D0
     . *P**4*M30*M10*J30X+9.0D0*P**2*M30*J30Y*J10Y+9.0D0*P**2
     . *M30*J30Y*J30X+9.0D0*P**2*M30*J10Y*J30X+P**2*J30Y*M10*
     . J30X+J30Y*J10Y*J30X)
      DO 25005 J=1,3
```

```
          DO 25006 K=J+1,3
              M(K,J)=M(J,K)
              MIV(K,J)=MIV(J,K)
25006     CONTINUE
25005 CONTINUE
```

Contents of file m2.f:

```
      M(1,1)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(DSIN(DBLE(
     . Q3))**2*Y*J30)+DSIN(DBLE(Q3))**2*J30Z+18.0D0*DCOS(DBLE
     . (Q3))*DCOS(DBLE(Q2))*P**2*M30+18.0D0*P**2*M30+P**2*M10
     . +J30Y+J10Y(1,2)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(
     . DSIN(DBLE(Q3))**2*J30Y)+DSIN(DBLE(Q3))**2*J30Z+9.0D0*
     . DCOS(DBLE(Q3))*DCOS(DBLE(Q2))*P**2*M30+9.0D0*P**2*M30+
     . J30Y(1,3)=-(9.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*P**2*
     . M30)
      M(2,2)=-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30)-(DSIN(DBLE(
     . Q3))**2*J30Y)+DSIN(DBLE(Q3))**2*J30Z+9.0D0*P**2*M30+
     . J30Y
      M(2,3)=0.0D0
      M(3,3)=9.0D0*P**2*M30+J30X
      T1=-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2)-(9.0D0*DSIN(
     . DBLE(Q3))**2*P**2*M30*J30Y)+9.0D0*DSIN(DBLE(Q3))**2*P
     . **2*M30*J30Z-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30X)-(
     . DSIN(DBLE(Q3))**2*J30Y*J30X)+DSIN(DBLE(Q3))**2*J30Z*
     . J30X+81.0D0*P**4*M30**2+9.0D0*P**2*M30*J30Y+9.0D0*P**2
     . *M30*J30X+J30Y*J30X
      T0=729.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30
     . **3+81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y-(81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**
     . 2*P**4*M30**2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*
     . M30**2*J30-(81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)
     . +9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(9.0D0*
     . DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Z*J30)
      T0=T0+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30X*J30-(
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+9.0D0*DSIN(
     . DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN(DBLE(Q3))
     . **4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*J30Y*J30X*
     . J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(DSIN(DBLE(Q3)
     . )**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y*J30Z*J30X-(
     . 729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**6*M30**
     . 3)
      T0=T0-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6*M30**3)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(81.0D0*DSIN
     . (DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*DSIN(DBLE(Q3))
```

```
     . **2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(Q3))**2*P**4*
     . M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*
     . J30X)
      T0=T0-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Y*M10)+9.0D0
     . *DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(9.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P
     . **2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*
     . M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y**2
     . -(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*J10Y)+9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y
      T0=T0+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))
     . **2*P**2*J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*
     . M10*J30X-(DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE
     . (Q3))**2*J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*
     . J30X)+DSIN(DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(
     . DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**6*M30**3)
      T0=T0-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*
     . M30**2*J30X)+729.0D0*P**6*M30**3+81.0D0*P**6*M30**2*
     . M10+81.0D0*P**4*M30**2*J30Y+81.0D0*P**4*M30**2*J10Y+
     . 81.0D0*P**4*M30**2*J30X+9.0D0*P**4*M30*J30Y*M10+9.0D0*
     . P**4*M30*M10*J30X+9.0D0*P**2*M30*J30Y*J10Y+9.0D0*P**2*
     ; M30*J30Y*J30X
      MIV(1,1)=T1/(T0+9.0D0*P**2*M30*J10Y*J30X+P**2*J30Y*M10*
     . J30X+J30Y*J10Y*J30X)
      T0=81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2+9.0D0*DSIN(DBLE
     . (Q3))**2*P**2*M30*J30Y-(9.0D0*DSIN(DBLE(Q3))**2*P**2*
     . M30*J30Z)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30X+DSIN(
     . DBLE(Q3))**2*J30Y*J30X-(DSIN(DBLE(Q3))**2*J30Z*J30X)-(
     . 81.0D0*DCOS(DBLE(Q3))*DCOS(DBLE(Q2))*P**4*M30**2)-(
     . 9.0D0*DCOS(DBLE(Q3))*DCOS(DBLE(Q2))*P**2*M30*J30X)-(
     . 81.0D0*P**4*M30**2)-(9.0D0*P**2*M30*J30Y)
      T1=729.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30
     . **3+81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y-(81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**
     . 2*P**4*M30**2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*
     . M30**2*J30-(81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)
     . +9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(9.0D0*
```

```
     . DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Z*J30)
      T1=T1+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30X*J30-(
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+9.0D0*DSIN(
     . DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN(DBLE(Q3))
     . **4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*J30Y*J30X*
     . J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(DSIN(DBLE(Q3)
     . )**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y*J30Z*J30X-(
     . 729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**6*M30**
     . 3)
      T1=T1-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6*M30**3)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(81.0D0*DSIN
     . (DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*DSIN(DBLE(Q3))
     . **2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(Q3))**2*P**4*
     . M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*
     . J30X)
      T1=T1-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Y*M10)+9.0D0
     . *DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(9.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P
     . **2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*
     . M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y**2
     . -(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*J10Y)+9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y
      T1=T1+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))
     . **2*P**2*J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*
     . M10*J30X-(DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE
     . (Q3))**2*J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*
     . J30X)+DSIN(DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(
     . DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**6*M30**3)
      T1=T1-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*
     . M30**2*J30X)+729.0D0*P**6*M30**3+81.0D0*P**6*M30**2*
     . M10+81.0D0*P**4*M30**2*J30Y+81.0D0*P**4*M30**2*J10Y+
     . 81.0D0*P**4*M30**2*J30X+9.0D0*P**4*M30*J30Y*M10+9.0D0*
     . P**4*M30*M10*J30X+9.0D0*P**2*M30*J30Y*J10Y+9.0D0*P**2*
     . M30*J30Y*J30X
      MIV(1,2)=(T0-(9.0D0*P**2*M30*J30X)-(J30Y*J30X))/(T1+
     . 9.0D0*P**2*M30*J10Y*J30X+P**2*J30Y*M10*J30X+J30Y*J10Y*
```

```
     . J30X)
      T0=729.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30
     . **3+81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y-(81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**
     . 2*P**4*M30**2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*
     . M30**2*J30-(81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)
     . +9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(9.0D0*
     . DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Z*J30)
      T0=T0+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30X*J30-(
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+9.0D0*DSIN(
     . DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN(DBLE(Q3))
     . **4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*J30Y*J30X*
     . J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(DSIN(DBLE(Q3)
     . )**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y*J30Z*J30X-(
     . 729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**6*M30**
     . 3)
      T0=T0-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6*M30**3)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(81.0D0*DSIN
     . (DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*DSIN(DBLE(Q3))
     . **2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(Q3))**2*P**4*
     . M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*
     . J30X)
      T0=T0-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Y*M10)+9.0D0
     . *DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(9.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P
     . **2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*
     . M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y**2
     . -(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*J10Y)+9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y
      T0=T0+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))
     . **2*P**2*J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*
     . M10*J30X-(DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE
     . (Q3))**2*J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*
     . J30X)+DSIN(DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(
     . DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**6*M30**3)
      T0=T0-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*
```

```
     . M30**2*J30X)+729.0D0*P**6*M30**3+81.0D0*P**6*M30**2*
     . M10+81.0D0*P**4*M30**2*J30Y+81.0D0*P**4*M30**2*J10Y+
     . 81.0D0*P**4*M30**2*J30X+9.0D0*P**4*M30*J30Y*M10+9.0D0*
     . P**4*M30*M10*J30X+9.0D0*P**2*M30*J30Y*J10Y+9.0D0*P**2*
     . M30*J30Y*J30X
      MIV(1,3)=(-(81.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**
     . 4*M30**2)-(9.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**2
     . *M30*J30Y)+9.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**2
     . *M30*J30Z+81.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*P**4*
     . M30**2+9.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*P**2*M30*
     . J30Y)/(T0+9.0D0*P**2*M30*J10Y*J30X+P**2*J30Y*M10*J30X+
     . J30Y*J10Y*J30X)
      T0=-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**4*
     . M30**2)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2)-(9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*Y*M30*J30)+9.0D0*DSIN(DBLE(Q3))
     . **2*P**2*M30*J30Z-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*
     . J30X)-(DSIN(DBLE(Q3))**2*Y*J30X*J30)+DSIN(DBLE(Q3))**2
     . *J30Z*J30X+162.0D0*DCOS(DBLE(Q3))*DCOS(DBLE(Q2))*P**4*
     . M30**2+18.0D0*DCOS(DBLE(Q3))*DCOS(DBLE(Q2))*P**2*M30*
     . J30X+162.0D0*P**4*M30**2
      T1=729.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30
     . **3+81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y-(81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**
     . 2*P**4*M30**2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*
     . M30**2*J30-(81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)
     . +9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(9.0D0*
     . DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Z*J30)
      T1=T1+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30X*J30-(
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+9.0D0*DSIN(
     . DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN(DBLE(Q3))
     . **4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*J30Y*J30X*
     . J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(DSIN(DBLE(Q3)
     . )**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y*J30Z*J30X-(
     . 729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**6*M30**
     . 3)
      T1=T1-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6*M30**3)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(81.0D0*DSIN
```

```
     . (DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*DSIN(DBLE(Q3))
     . **2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(Q3))**2*P**4*
     . M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*
     . J30X)
      T1=T1-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Y*M10)+9.0D0
     . *DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(9.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P
     . **2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*
     . M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y**2
     . -(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*J10Y)+9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y
      T1=T1+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))
     . **2*P**2*J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*
     . M10*J30X-(DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE
     . (Q3))**2*J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*
     . J30X)+DSIN(DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(
     . DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**6*M30**3)
      T1=T1-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*
     . M30**2*J30X)+729.0D0*P**6*M30**3+81.0D0*P**6*M30**2*
     . M10+81.0D0*P**4*M30**2*J30Y+81.0D0*P**4*M30**2*J10Y+
     . 81.0D0*P**4*M30**2*J30X+9.0D0*P**4*M30*J30Y*M10+9.0D0*
     . P**4*M30*M10*J30X+9.0D0*P**2*M30*J30Y*J10Y+9.0D0*P**2*
     . M30*J30Y*J30X
      MIV(2,2)=(T0+9.0D0*P**4*M30*M10+9.0D0*P**2*M30*J30Y+
     . 9.0D0*P**2*M30*J10Y+18.0D0*P**2*M30*J30X+P**2*M10*J30X
     . +J30Y*J30X+J10Y*J30X)/(T1+9.0D0*P**2*M30*J10Y*J30X+P**
     . 2*J30Y*M10*J30X+J30Y*J10Y*J30X)
      T0=729.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30
     . **3+81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y-(81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**
     . 2*P**4*M30**2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*
     . M30**2*J30-(81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)
     . +9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(9.0D0*
     . DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Z*J30)
      T0=T0+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30X*J30-(
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+9.0D0*DSIN(
     . DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN(DBLE(Q3))
```

```
     . **4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*J30Y*J30X*
     . J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(DSIN(DBLE(Q3)
     . )**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y*J30Z*J30X-(
     . 729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**6*M30**
     . 3)
      T0=T0-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6*M30**3)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(81.0D0*DSIN
     . (DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*DSIN(DBLE(Q3))
     . **2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(Q3))**2*P**4*
     . M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*
     . J30X)
      T0=T0-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Y*M10)+9.0D0
     . *DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(9.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P
     . **2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*
     . M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y**2
     . -(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*J10Y)+9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y
      T0=T0+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))
     . **2*P**2*J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*
     . M10*J30X-(DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE
     . (Q3))**2*J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*
     . J30X)+DSIN(DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(
     . DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**6*M30**3)
      T0=T0-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*
     . M30**2*J30X)+729.0D0*P**6*M30**3+81.0D0*P**6*M30**2*
     . M10+81.0D0*P**4*M30**2*J30Y+81.0D0*P**4*M30**2*J10Y+
     . 81.0D0*P**4*M30**2*J30X+9.0D0*P**4*M30*J30Y*M10+9.0D0*
     . P**4*M30*M10*J30X+9.0D0*P**2*M30*J30Y*J10Y+9.0D0*P**2*
     . M30*J30Y*J30X
      MIV(2,3)=(81.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**4*
     . M30**2+9.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**2*M30
     . *J30Y-(9.0D0*DSIN(DBLE(Q3))**3*DSIN(DBLE(Q2))*P**2*M30
     . *J30Z)-(81.0D0*DSIN(DBLE(Q3))*DSIN(DBLE(Q2))*DCOS(DBLE
     . (Q3))*DCOS(DBLE(Q2))*P**4*M30**2)-(81.0D0*DSIN(DBLE(Q3
     . ))*DSIN(DBLE(Q2))*P**4*M30**2)-(9.0D0*DSIN(DBLE(Q3))*
```

```
     . DSIN(DBLE(Q2))*P**2*M30*J30Y))/(T0+9.0D0*P**2*M30*J10Y
     . *J30X+P**2*J30Y*M10*J30X+J30Y*J10Y*J30X)
      T0=9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30-(9.0D0*DSIN(
     . DBLE(Q3))**4*P**2*M30*J30Y)+DSIN(DBLE(Q3))**4*Y*J30Y*
     . J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30)-(DSIN(DBLE(Q3))**4*
     . J30Y**2)+DSIN(DBLE(Q3))**4*J30Y*J30Z-(81.0D0*DSIN(DBLE
     . (Q3))**2*P**4*M30**2)-(9.0D0*DSIN(DBLE(Q3))**2*P**4*
     . M30*M10)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*M30*J30)+
     . 9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z
      T0=T0-(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J10Y)-(DSIN(
     . DBLE(Q3))**2*P**2*J30Y*M10)+DSIN(DBLE(Q3))**2*P**2*
     . J30Z*M10-(DSIN(DBLE(Q3))**2*Y*J30Y*J30)+DSIN(DBLE(Q3))
     . **2*J30Y**2-(DSIN(DBLE(Q3))**2*J30Y*J10Y)+DSIN(DBLE(Q3
     . ))**2*J30Z*J10Y-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2
     . ))**2*P**4*M30**2)+81.0D0*P**4*M30**2+9.0D0*P**4*M30*
     . M10+9.0D0*P**2*M30*J30Y
      T1=729.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**6*M30
     . **3+81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y-(81.0D0*DSIN(DBLE(Q3))**4*DSIN(DBLE(Q2))**
     . 2*P**4*M30**2*J30Z)+81.0D0*DSIN(DBLE(Q3))**4*P**4*Y*
     . M30**2*J30-(81.0D0*DSIN(DBLE(Q3))**4*P**4*M30**2*J30Y)
     . +9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Y*J30-(9.0D0*
     . DSIN(DBLE(Q3))**4*P**2*Y*M30*J30Z*J30)
      T1=T1+9.0D0*DSIN(DBLE(Q3))**4*P**2*Y*M30*J30X*J30-(
     . 9.0D0*DSIN(DBLE(Q3))**4*P**2*M30*J30Y**2)+9.0D0*DSIN(
     . DBLE(Q3))**4*P**2*M30*J30Y*J30Z-(9.0D0*DSIN(DBLE(Q3))
     . **4*P**2*M30*J30Y*J30X)+DSIN(DBLE(Q3))**4*Y*J30Y*J30X*
     . J30-(DSIN(DBLE(Q3))**4*Y*J30Z*J30X*J30)-(DSIN(DBLE(Q3)
     . )**4*J30Y**2*J30X)+DSIN(DBLE(Q3))**4*J30Y*J30Z*J30X-(
     . 729.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**6*M30**
     . 3)
      T1=T1-(81.0D0*DSIN(DBLE(Q3))**2*DSIN(DBLE(Q2))**2*P**4*
     . M30**2*J30Y)-(729.0D0*DSIN(DBLE(Q3))**2*P**6*M30**3)-(
     . 81.0D0*DSIN(DBLE(Q3))**2*P**6*M30**2*M10)-(81.0D0*DSIN
     . (DBLE(Q3))**2*P**4*Y*M30**2*J30)+81.0D0*DSIN(DBLE(Q3))
     . **2*P**4*M30**2*J30Z-(81.0D0*DSIN(DBLE(Q3))**2*P**4*
     . M30**2*J10Y)-(81.0D0*DSIN(DBLE(Q3))**2*P**4*M30**2*
     . J30X)
```

```
      T1=T1-(9.0D0*DSIN(DBLE(Q3))**2*P**4*M30*J30Y*M10)+9.0D0
     . *DSIN(DBLE(Q3))**2*P**4*M30*J30Z*M10-(9.0D0*DSIN(DBLE(
     . Q3))**2*P**4*M30*M10*J30X)-(9.0D0*DSIN(DBLE(Q3))**2*P
     . **2*Y*M30*J30Y*J30)-(9.0D0*DSIN(DBLE(Q3))**2*P**2*Y*
     . M30*J30X*J30)+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y**2
     . -(9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Y*J10Y)+9.0D0*
     . DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J10Y
      T1=T1+9.0D0*DSIN(DBLE(Q3))**2*P**2*M30*J30Z*J30X-(9.0D0
     . *DSIN(DBLE(Q3))**2*P**2*M30*J10Y*J30X)-(DSIN(DBLE(Q3))
     . **2*P**2*J30Y*M10*J30X)+DSIN(DBLE(Q3))**2*P**2*J30Z*
     . M10*J30X-(DSIN(DBLE(Q3))**2*Y*J30Y*J30X*J30)+DSIN(DBLE
     . (Q3))**2*J30Y**2*J30X-(DSIN(DBLE(Q3))**2*J30Y*J10Y*
     . J30X)+DSIN(DBLE(Q3))**2*J30Z*J10Y*J30X-(729.0D0*DCOS(
     . DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**6*M30**3)
      T1=T1-(81.0D0*DCOS(DBLE(Q3))**2*DCOS(DBLE(Q2))**2*P**4*
     . M30**2*J30X)+729.0D0*P**6*M30**3+81.0D0*P**6*M30**2*
     . M10+81.0D0*P**4*M30**2*J30Y+81.0D0*P**4*M30**2*J10Y+
     . 81.0D0*P**4*M30**2*J30X+9.0D0*P**4*M30*J30Y*M10+9.0D0*
     . P**4*M30*M10*J30X+9.0D0*P**2*M30*J30Y*J10Y+9.0D0*P**2*
     . M30*J30Y*J30X
      MIV(3,3)=(T0+9.0D0*P**2*M30*J10Y+P**2*J30Y*M10+J30Y*
     . J10Y)/(T1+9.0D0*P**2*M30*J10Y*J30X+P**2*J30Y*M10*J30X+
     . J30Y*J10Y*J30X)
      DO 25007 J=1,3
          DO 25008 K=J+1,3
              M(K,J)=M(J,K)
              MIV(K,J)=MIV(J,K)
25008     CONTINUE
25007 CONTINUE
```

References