SLAC COMPUTATION GROUP
Stanford, California

CGTM No. 206
August 1981

Using Mortran on VM

Roger B. Chaffee
Computation Research Group
Stanford Linear Accelerator Center

CONTENTS

Chapter 1

MORTRAN 2.79

I have adapted the Mortran processor to my own needs. This note describes my version, which is running on SLAC's VM system, and has been sent out with Top Drawer. Most of the code is the same as Jim Cook's Mortran, and much of this note has been copied from Jim's technical memos.

The primary guide to using Mortran is A User's Guide to MORTRAN2, by Jim Cook and Len Shustek, CGTM No. 165. Read that one first.

1.1 MORTRAN SYNTAX

With a few exceptions involving strings, any Fortran statement, followed by a semicolon, is sent to the output file. No checking is done by the Mortran processor that the statements are in fact legal Fortran.
Exceptions:

1. Strings within apostrophes (e.g. 'string') are turned into Hollerith form.
2. Multiple blanks in the input, not within apostrophes, are turned into a single blank. This includes blanks within a Hollerith string.

The above statements about the Mortran processor result in the following general rules about writing Mortran code:

1. Statements are free-form, terminated by a semicolon (;). Card boundaries are irrelevant.
2. Comments are enclosed in quotation marks ("comment") and may appear anywhere.
3. Character strings are enclosed in apostrophes ('string').
4. Alphanumeric statements labels are enclosed in colons (:label:) and may appear anywhere a Fortran statement label is legal.

1.2 MORTRAN STATEMENTS

In the following, "stmts;" means zero or more Fortran statements, each followed by a semicolon, and "loop" means WHILE-loop, UNTIL-loop, BLOCK, LOOP, FOR-loop or DO-loop.

1.2.1 Conditionals

```
IF (lexpr) stmt;        (This, of course, is Fortran.)
IF lexpr <stmts;>
IF lexpr <stmts;> ELSE <stmts;>
IF lexpr <stmts;> ELSEIF lexpr <stmts;> ELSE <stmts;>
```

1.2.2 Loops

```
WHILE lexpr < stmts; >
UNTIL lexpr < stmts; >
BLOCK       < stmts; >
LOOP        < stmts; >
LOOP        < stmts; > UNTIL lexpr;
LOOP        < stmts; > WHILE lexpr;
LOOP        < stmts; > REPEAT;
FOR v=e BY e TO e < stmts; >
FOR v=e TO e BY e < stmts; >
FOR v=e TO e      < stmts; >
DO  v=e,e,e       < stmts; >    (Fortran DO-loop.)
DO  v=e,e,e       < stmts; > FALL THRU <stmts;>
<v=e,e,e; stmts;>               (Fortran DO-loop.)
<v=e,e,e; stmts;> FALL THRU <stmts;>
```

These loops may all be nested. E.g.

```
<I=1,200;
   J=I; UNTIL M(J).EQ.0 < J=M(J);>
   IF I.NE.J <
      OUTPUT I,M(J+1); (' Chain',I4,' ends with ',A4);
>  >
```

Within one of these loops, the statement EXIT; means "branch to the statement following the end of the loop". The statement NEXT; means "start at the beginning of the loop". (For DO-loops and FOR-loops, the counter is incremented.) E.g.

```
LOOP < INPUT X; (F10.0); IF X.EQ.0.0 EXIT; SUM=SUM+X;>
```

A loop may be labeled. A NEXT or EXIT statement may refer to the loop at any nesting level above it by using the label. E.g.

```
:FIND BLANK:
LOOP <
   INPUT CARD; (80A1);
   <I=1,80; IF (CARD(I).EQ.BLANK) EXIT :FIND BLANK:; >
>
```

Note that there must not be a semicolon between the label and the following keyword. (This would cause a labelled 'continue' statement, but would not associate the label with the loop.)

1.2.3 Local Subroutines

Within a Fortran subroutine, local procedures can be defined with the following syntax:

```
:procname: PROCEDURE < stmts; >
```

A procedure can be invoked by the statement

```
EXECUTE  :procname:;
```

Within the body of the procedure, the statement EXIT :procname:; effects a 'return' to the statement following the EXECUTE statement, as does the final '>'.

Returns from local procedures are implemented with an ASSIGN statement and an assigned GOTO, which are legal but unusual constructs. The 'call' is done with a GOTO, which is always legal in Fortran IV. However, the new Fortran 77 explicitly prohibits branching out of the range of a DO-loop and then returning, so this construct will produce illegal code if an EXECUTE statement inside a DO-loop calls a PROCEDURE outside it.

1.2.4 Abbreviations

1.2.4.1 Multiple Assignment

```
/v1,v2,...,vn/ = e;
Resulting Fortran:
      v1=e
      v2=e
       ...
      vn=e
```

1.2.4.2 INPUT and OUTPUT Statements

```
OUTPUT; (fmtlist);
OUTPUT varlist; (fmtlist);
INPUT varlist; (fmtlist);
```

Resulting Fortran:

```
          WRITE (6,20)
20        FORMAT (fmtlist)

          WRITE (6,30) varlist
30        FORMAT (fmtlist)

          READ (5,40) varlist
40        FORMAT (fmtlist)
```

1.3 QUOTED STRINGS

%Vn in columns 1 to 3 controls the treatment of quoted strings.

%V0 is the default setting, which causes strings within apostrophes to be converted to equivalent Hollerith values. A double apostrophe, i.e. a null string, is changed to a single apostrophe.

%V1 causes strings within apostrophes to be passed along unchanged. In this case, there is no way to produce an unpaired apostrophe in the output.

1.4 COMMENTS

%K in columns 1 and 2 of a card causes that entire card to be copied as a comment card in the Fortran output.

%Ynn starting in column 1 causes Mortran comments, enclosed in quotes in the input, to appear in comment cards in the output. The comment text will start in column nn+1 of the card, where nn is a 1- or 2-digit unsigned decimal integer. nn=0 or blank inhibits this feature. No more than one comment card will be produced between normal output lines, so large blocks of comments will not be passed.

The internal operator @NN performs the same function for the Fortran output that the @NM operator performs for the listing, that is, the first actual parameter in the macro being expanded is written to the file in question. (In CGTM No. 185 and No. 167, the @NM operator is incorrectly given as @MM. The original intent may have been to use @MM, but all implementations I have seen have used @NM, as does my version.)

1.5 CONDITIONAL GENERATION OF CODE

In addition to the usual GENERATE and NOGENERATE control words, I have added an ELSEGENERATE. For example,

```
GENERATE; A=B; ELSEGENERATE; C=D; ENDGENERATE;
```

produces the code line A=B, but

```
NOGENERATE; A=B; ELSEGENERATE; C=D; ENDGENERATE;
```

produces C=D. All GENERATE controls can be nested in the usual way.

1.6 LISTING

There are three levels of listing possible. Level 0 gives no listing at all. Level 1 lists only the %E lines, which are normally page ejects and can be used for comments. Level 2 is the full listing. %Ln or %Nn in columns 1 through 3 sets the listing level to n. %L is the same as %L2, and %N is the same as %L1.

1.7 MORTRAN KEYWORDS

Mortran programs may be written using either 'reserved-word' or 'bracketed' keywords. A program using reserved-word keywords might look like this:

```
FOR I=1 TO 8
   <J=I**2; OUTPUT I,J; (' I=',I2,'J=',I4); >
```

The equivalent program, using bracketed keywords, is

```
<FOR> I=1 <TO> 8
   <J=I**2; <OUTPUT> I,J; (' I=',I2,'J=',I4); >
```

The choice of macros is one of style. Most people seem to use the reserved-word macros, perhaps because they are easier to type, or because they look more like Algol or Pascal. My own preference is for the bracketed keywords, because the code seems easier to read that way. I don't know of any fundamental reason to prefer one to the other. I suppose it would be possible to allow both macro sets in one set of input, but Iit would be a complex process to implement it.

The examples in this note all use the reserved-word set. To use the bracketed set, simply enclose each keyword in 'French quotes', i.e. use <LOOP> instead of LOOP, <DO> instead of DO, etc.

The specification of macros depends on the particular system. Each macro set is in a different input file, and one or the other must be read in by the processor at execution time. Since Mortran is invoked in different ways on different systems, I can't tell you here how to use it on your system.

1.8 <RESET>

The characters '<SET>' appearing in the input cause a pointer to be added to a stack which saves the current state of the macros. The characters '<RESET>' pop that stack and restore the macros to the state that existed when the corresponding <SET> command was encountered.

1.9 COMPILE-TIME ARITHMETIC

Mortran has an arithmetic stack, and allows manipulation of the contents of that stack through commands which can be imbedded in the input code. This feature is somewhat esoteric, but can be useful for writing simple macros, e.g. for funny dimension statements.

nnnn<ENT> push the integer nnnn onto the top of the stack. nnnn is a decimal integer.

<VAL> pop the number on the top of the stack into the code stream. (It is in the same format as other Mortran-generated numbers, i.e. as the two characters @n, and is not translated to a decimal-character representation until it is moved to the output buffer.)

<DEF>name; Create a macro pattern 'name', and use the number on top of the stack as the replacement part. This number is popped, i.e. removed, from the stack. Note that the semicolon is required, but is not part of 'name' and is not put in the output stream. (<DEF> differs from <VAL> in that macros involving <VAL> usually pop a new number off the stack each time they are expanded, and <DEF> pops the stack once, when 'name' is defined as a macro pattern.)

<+>
<->
<*>
</> perform the indicated operation between the second number on the stack and the top of the stack, pop both those numbers from the stack and push the result onto the stack. Division by zero is not performed, but no other checks are made on the values used.

<DUP> Duplicate the top of the stack. (The stack becomes one value longer, and the top two are the same.)

<EQ>
<NE>
<GT>
<GE>
<LT>
<LE>

The value on top of the stack is removed from the stack and compared to zero. If the comparison is satisfied, a <GENERATE> macro is generated. If not, a <NOGENERATE> macro is generated.

<END>

This is the same as <ENDGENERATE>.

<SWAPn>

For n=1,...,9, this swaps the value at the top of the stack (the zero-th value) with the n-th value in the stack. For instance, <SWAP1> exchanges the top two values in the stack.

%S1 appearing in columns 1-3 is a debugging aid which will cause the printing of the stack contents each time an arithmetic stack operation is done.

1.9.1 Example

Input

```
%'$BUFSIZ'='500'
$BUFSIZ<ENT>2<ENT><*>
DIMENSION X(<VAL>);
$BUFSIZ<ENT>999<ENT>
<->><DUP>
<GE>BUFSIZ GE999; XXXXXX= <END>
<LE>BUFSIZ LE999; YYYY= <END>
-15 <ENT> 4 <ENT> <+> <VAL>;
%S1
%'$BUFSIZ'='5000'
DIMENSION X($BUFSIZ<ENT>2<ENT><*><VAL>);
$BUFSIZ<ENT>999<ENT>
<->><DUP>
<GE>BUFSIZ GE999; XXXXXX= <END>
<LE>BUFSIZ LE999; YYYY= <END>
-15 <ENT> 4 <ENT> <+> <VAL>;
%%
```

Printer Output

```
        Mortran 2.0     (VERSION OF 4/01/79)

  107      0    %U5
  108      0    %'$BUFSIZ'='500'
  109      0    $BUFSIZ<ENT>2<ENT></>
  110      0    DIMENSION X(<VAL>);
  111      0    $BUFSIZ<ENT>999<ENT>
  112      0    <->DUP>
  113      0    <GE>BUFSIZ GE999; XXXXXX= <END>
  114      0    <LE>BUFSIZ LE999; YYYY= <END>
  115      0    -15 <ENT> 4 <ENT> <+> <VAL>;
  116      0    %S1
  117      0    %'$BUFSIZ'='5000'
  118      0    DIMENSION X($BUFSIZ<ENT>2<ENT></><VAL>);
ARITH STACK       0   5000
ARITH STACK       0   5000      2
ARITH STACK       0   2500
ARITH STACK       0
  119      0    $BUFSIZ<ENT>999<ENT>
ARITH STACK       0   5000
ARITH STACK       0   5000    999
  120      0    <->DUP>
ARITH STACK       0   4001
ARITH STACK       0   4001   4001
  121      0    <GE>BUFSIZ GE999; XXXXXX= <END>
  122      0    <LE>BUFSIZ LE999; YYYY= <END>
  123      0    -15 <ENT> 4 <ENT> <+> <VAL>;
ARITH STACK       0    -15
ARITH STACK       0    -15      4
ARITH STACK       0    -11
ARITH STACK       0
  124      0    %%
    0 Mortran ERRORS ENCOUNTERED
```

Fortran Output

```
C     Mortran 2.0     (VERSION OF 4/01/79)
      DIMENSION X(250)
      BUFSIZ LE999
      YYYY=      -11
      DIMENSION X(2500)
      BUFSIZ GE999
      XXXXXX=       -11
```

Chapter 2

USING MACROS

The only guide to using the full power of the Mortran processor is C. T. Zahn's A User Manual for the MORTRAN2 Macro-translator, CGTM No. 167. Zahn's SKOL language, implemented in Mortran macros, is a demonstration of the remarkable power of Mortran.

Some of the physics groups at SLAC have developed macro sets for common programming applications, using the standard macros as a starting point and sample. 'Mortran Macro' is a difficult language to program in, but the possibility of extending the language to match a particular application is to my knowledge unmatched in a Fortran environment. Appendix B, which is mostly copied from Zahn, gives a list of the @-operators currently available. It should be emphasized that these operators are processed when they are found in the replacement part of a macro-rule, when that rule is found to match a pattern in the code. An '@' in the code itself has no significance.

Chapter 3

USING MORTRAN ON SLAC VM

Mortran is on the system U-disk, and is available through the Mortran command:

MORTRAN [fn|fn ft|fn ft fm] [([RES|BRK] [ONTO filemode]]

3.1 SPECIFYING THE MACRO SET

The choice of macros is one of style. You may specify it in several ways:

1) An input file which has a filetype of MORTRANR requires the reserved-word macros. Similarly, MORTRANB requires bracketed-keyword macros.

2) For any other filetype, you may use BRK or RES as a parameter when you invoke the exec.
   E.g. MORTRAN NEW PROGRAM (BRK

3) The FIRST line in a file named -name- MORTDEF (where -name- is the filename of your program) may be either BRK or RES, which will invoke the corresponding macro set if it has not been specified by any of the above.

4) The FIRST line in a file named PROFILE MORTDEF may be either BRK or RES, which will invoke the corresponding macro set if it has not been specified by any of the above.

3.2 OUTPUT FILES

MORTRAN writes three files:

Unit	OS equivalent	Filetype	Contents
4	SYSTERM		Error messages and %E lines
6	SYSPRINT	MORTLIST	Full listing.
7	SYSPUNCH	FORTRAN	Fortran cards.

The SYSTERM output is normally written directly to the terminal, so there is no default file name. The other two are disk files. The filename for both is the program name, which you use in the MORTRAN statement. The filetypes are MORTLIST and FORTRAN.

The output disk can be set in one of four ways.

1) Explicitly, by an 'ONTO diskmode' parameter in the invocation, e.g. MORTRAN XXX (ONTO A5)

2) If not set explicitly, the disk mode is set to the same value as that for the input file.

3) If neither of these points to a disk for which you have write access, then the disk mode is set to 'A1'.

4) Finally, a FILEDEF command in your 'PROFILE MORTDEF' or 'name MORTDEF' file overrides any of the above settings, and can be used to set filename, filetype, and disk for any Mortran input or output file.

3.3 FILEDEFS FOR ADDITIONAL FILES

When MORTRAN is executed on a file named 'name', first the default input and output files are FILEDEF'd. Then your disks are searched for a file named PROFILE MORTDEF, which could contain additional FILEDEF commands. With a PROFILE MORTDEF file you can set up FILEDEFs for the files that your Mortran programs normally require. Finally, the same search and read is performed using the filename 'name MORTDEF', so you can set up FILEDEFs which are specific to the particular 'name' you are working with.

The MORTDEF files should be in CARD (F 80) format, or some other format which can be read by the DISKIO program. This is a consideration only if you create them from WYLBUR.

For instance, if you have a set of lines which you want to include in many places in your program, the following arrangement will work:

The file NAME COMMON A contains lines to be inserted.
The file NAME MORTDEF A contains the line

```
FILEDEF 22 DISK NAME COMMON A
```

The input, in the file NAME MORTRAN A, contains the lines

```
%R22
%U22
```

where ever the insertion is to occur.

Mortran is then invoked by the statement MORTRAN NAME. (Of course, for inserting up to a few tens of lines, it would be better to create a macro, such as

```
%'INSERTION'='lines or statements to be inserted'
```

In this case, the statement INSERTION; in the source will be replaced by the lines to be inserted, without the I/O activity.)

Appendix A

CONTROL COMMANDS

For reference, these are the commands for program control, all of which start with '%' in column 1, and have a key symbol in column 2. These commands are processed before they reach the normal input stream, which means that they do not affect the other input, and also that one cannot create a macro which produces the same result.

```
%An  Annotation
       n=0: Source not put in Fortran output
         1: Source starts in col. 2 of Fortran comment cards
         2: Source starts in col. 41 of Fortran comment cards

%Cnn Card length. nn from 10 to 80.

%Dn  Macro Definitions Trace
        n=0: Trace Off.
        n=1,2: Trace On.

%E   Page Eject.
        New page is started.  (Entire card is listed.)

%F   Fortran mode.  (See %M.)
         Input lines are copied to the output, until a %M.

%Inn Indentation.
         Lines are listed with indentation of nn*L spaces,
         where L is the current nesting level.

%K   Komment Kopy
        The entire card is passed as a comment card to the
        Fortran output.

%Ln  Listing control.  (See %N.)
       n=0: No listing.
         1: List only %E lines.
         2: Full listing.
         blank: same as %L2
```

```
%M   Enter Mortran mode.  (See %F.)

%Nn  Disable Listing.  (See %L.)
       n=0: No listing.
         1: List only %E lines.
         2: Full listing.
         blank: same as %N1

%Onn Output Unit.
        nn=1,99 sets the Fortran unit for normal listing.

%Qn  Comment Termination
          n=0: Comments must be terminated explicitly.
            1: Comments end at the end of line, regardless.

%Rnn REWIND Fortran unit nn.

%Sn  Arithmetic Stack Trace.
        n=0: Trace off.
          1: Trace on.
          2: Internal gut-dumping time.

%Tn  Macro Recognition Trace
        n=0: Trace off.
          1: Trace selected macros. (Those containing '@MT')
          2: Full trace. (Listing for each pattern matched.)

%Unn Unit.
        Start reading from Fortran Unit nn.

%U filename filetype filemode [etc.] (SLAC VM version only.)
        A command is issued using the KMAND subroutine,
          with 'FILEDEF nn DISK' followed by whatever text
          is on the %U card. nn=99 for the first such card,
          followed by 98, 97, etc., for subsequent cards.
          Mortran then starts reading from unit nn.

%Vn  Hollerith Conversion.
        n=0: Quoted strings converted to Hollerith.
          1: Quoted strings passed without conversion.

%Ynn Comments in Fortran Output.
        n=0: Comments in source are ignored.
        n=1,78: Comments in source are passed as comments in
          Fortran output, with text starting in column nn.
          (No more than one comment card is generated
          between Fortran statements.)

%Znn Error Message Unit.   ('SYSTERM')
        nn=1,99 sets the Fortran unit number for error
        messages and %E lines.  This is useful in an
        interactive environment.
```

%% End of Input Unit
    Reading resumes from previous input unit. (Current unit is not rewound, and could be continued.)

%%R End of Input Unit
    Reading resumes from previous input unit. (Current unit is rewound. On some machines, this will release buffer space, with a consequent saving of system resources.)

Appendix B

OPERATORS IN REPLACEMENT PARTS OF MACROS

Operator	Subs.Text	Side Effect
@LG	@n	n is a unique integer > 10000 and divisible by 10.
@n@LSd	-	integer n is pushed into the label stack at position indicated by character decimal digit d.
@LUd	-	integer in label stack at position d is popped and discarded.
@MSx	-	character x is pushed onto the symbol stack.
@MU	@x	character at top of symbol stack ('x') is popped.
@MRx	-	character x replaces current symbol on top of symbol stack.
@MT	-	marks this macro rule for selective tracing whenever it is matched while selective tracing mode is in effect. (See %T in Appendix A.)
@I0	-	internal integer counter set to zero.
@I+	-	internal integer counter increased by 1.
@I-	-	internal integer counter decreased by 1.
@IC	@n	n is the current value of the integer counter
@n@I=	-	integer counter set to n.
@MG	-	generate macro-rule #1 --> #2
@MC	-	change existing macro #1--><rep> to #1-->rep#2. If no existing macro, then create new macro #1-->#2.

@MD	-	Most recent macro-rule for #1 is deleted. (Buffer space is not reclaimed)
@CN	-	'nogenerate' performed
@CG	-	'generate' performed
@CE	-	'endgenerate' performed
@CL	-	'elsegenerate' performed
@N+	-	nest-level counter increased by 1
@N-	-	nest-level counter decreased by 1
@NL	@n	n is card number
@NM	-	copy of text string matching first # in macro pattern is printed as a line of the source listing.
@NN	-	copy of text string matching first # in macro pattern is sent as a a line of the Fortran output.
@MF	-	current macro-rule status is stacked in the 'reset' stack.
@MM	-	'reset' stack is popped and macro status restored.
@n@LT	ccccc	ccccc is the 5-column character representation of the number n.
@LCij	@n	n is an integer equal to the sum of decimal digit j and the integer in the label stack at position indicated by decimal digit i.

The following description of the arithmetic stack assumes that the stack starts with the 4 values a b c d on it. The top of the stack, i.e. the last number pushed, is on the left.

Operator	Resulting Stack	Comment
@A+	(a+b) c d	
@A-	(b-a) c d	
@A*	(b*a) c d	
@A/	(b/a) c d	result is b if a=0
j@AQ	[j] b c...[j-1] a [j+1]	swap (top is [0])
@AR	a a b c d	dup
ccccc@AS	v a b c d	v is the integer given by the signed decimal characters ccccc
@AT	b c d	@a is put into the text stream
@AU	b c d	if a=0, perform @CG, otherwise @CN
@AV	b c d	if a is non-zero, perform @CG, otherwise @CN
@AW	b c d	if a<0, perform @CG, otherwise @CN
@AX	b c d	if a<=0, perform @CG, otherwise @CN
@AY	b c d	if a>=0, perform @CG, otherwise @CN
@AZ	b c d	if a>0, perform @CG, otherwise @CN