A FORTRAN TO MODULA 2 DICTIONARY

by

W.R. Knight
D.M. Fellows

TR92-073 September 1992

a

FORTRAN

to

[m][o][d][u][l][a] [2]

dictionary

W.R. Knight September 1990
D.M. Fellows January 1991
W.R. Knight September 1992

SOME DIFFERENCES FROM FORTRAN

Starting a new line does not start a new statement; statements end with a semicolon, except for the very last END statement which terminates the program: That ends with a period.

Modula distinguishes lower and upper case. Variables "CAT" and "cat" are not the same.

All Modula key words are in capitals.

All variables must be declared.

All main program variables are known to any procedure (suboutine). You must declare separately in the procedure to avoid changing main program variables.

Arguments of a procedure (subroutine) are declared in the argument list rather than with the other variables.

Modula does not allow mixed arithmetic, i.e. integer and real constants or variables in the same expression. (What is worse it will not always allow integer and cardinal in the same expression.)

NASTY ERROR SITUATIONS

In Stony Brook QuickMod, An error message insisting an END statement is missing probably means that the terminal END statement has gotten buried in an unclosed comment or in unclosed quotes, and this can be the hard to find. The problem can even be a comment opened inside of another comment.

FORTRAN (or Topic)	MODULA (or information)
( )	( ) to control order of computation and to enclose a procedure argument list [ ] for array indexing
=	:= The modula = is the fortran .EQ.
+	+
-	-
*	*
/	/ real divide DIV integer or cardinal divide
**	Not available, use pow from module MathLib0
ABS	ABS
AMAX1(X,Y) AMIN1(X,Y)	IF X>Y THEN Z:=X ELSE Z:=Y; IF X<Y THEN Z:=X ELSE Z:=Y;
ALOG	ln (lower case) import from MathLib0.
assignment	Modula uses := for assignment instead of = Assignment may convert types: For integer or cardinal to real use the built-in function FLOAT, for real to integer, use function entier from module MathLib0.
.AND.	AND The ampersand, & ,is also accepted.
ATAN	arctan (lower case) import from MathLib0
C	See Comment
case	The nifty CASE construction does not exist in Fortran. Example: VAR CaseSelector INTEGER (* or cardinal *); CASE CaseSelector OF 1: (* some computation. *) | 2..5: (* some computation. *) | 12: (* some computation *) | ELSE (* some computation *) END (*case*); As you would expect, the block of computation appropriate to the case selector is done. The case selector may be character (or an ennumerated type.) Cases are separated by a vertical bar.

CHARACTER LETTER CHARACTER WORD(6) CHARACTER WORD(0:5) CHARACTER WORD*6	VAR LETTER : CHAR VAR WORD : ARRAY [1..6] OF CHAR VAR WORD : ARRAY [0..5] OF CHAR VAR WORD : ARRAY [0..5] OF CHAR However, be careful; these are different: VAR CC : CHAR VAR CC : ARRAY [1..1] OF CHAR
character literals	Either single or double quotes may be used. LETTER:='A'; WORD:='CAT'; LETTER:="A"; WORD:="CAT";
COMMON	Put common variables and constants in a module and import the module.
Comment	In Modula a comment begins (* and ends *) A comment may occur anywhere so long as it doesn't break up a word. {Note: In QuickMod a (* inside of a comment makes trouble.
COMPLEX	Not included. It can't even be faked gracefully.
constant	See also PARAMETER. As in Fortran a decimal point specifies a real number. Modula does not distinguish between real and long real constants; the "D0" you have to keep tacking on in Fortran is neither allowed nor necessary. Quotes, double " or single ', are used for character constants.
CONTINUE	Not needed. For loop termination use END.
continuation in col 6	Irrelevant: Statements are separated by semicolons.
COS	cos (lower case) import from MathLib0.
DATA	Not available in vanilla Modula. You can set single constants using CONST (See PARAMETER). With arrays you need an assignment statement for every element, though you can hide this in a module created for the purpose. Stony Brook Modula has vector constants, see its manual.
/ divide	Real division, X / Y as of old. Integer orcardinal division, I DIV J (Modula disallows mixed mode arithmetic.)

DO .. I = 1,5

```
FOR I := 1 TO 5 DO
    (* statements *)
END;
```

DO .. I = 3,9,2

```
FOR I := 3 TO 9 BY 2 DO (* statements *) END;
                  Modula allows backward loops.
```

Note: No statement number ends a loop, the END keyword does that.
Note: Modula is fussy about types here and won't let you to mix integer and cardinal.

DOUBLE PRECISION X

```
VAR X:LONGREAL;       (* See also REAL*8 *)
```

(Stony Brook Modula supplies a module named LongMathLib0 paralleling the REAL module MathLib0.)

DIMENSION

```
ARRAY [lower_bound .. upper_bound] OF type
```

See also particular types, e.g. INTEGER, REAL,...

E

```
E                    examples:    2.0E3   -0.44E-5
```

ELSE

```
ELSE                                    See also IF
```

END

```
(* end of a procedure named PP      *)   END PP;
(* ends of IF, FOR, etc blocks      *)   END;
(* end of an entire module, MM,     *)
(* ends with a period not semiclon *)   END MM.
```

equal
.EQ.

Modula uses := for assignment, (see also) and = for Boolean expressions in IF statements, etc.

EQUIVALENCE

Use variant records or pointers. An example of each follows for printing the bit pattern of real 1.0/3.0 as a long cardinal.

```
(** example 1:  VARIANT RECORDS ************)
VAR   Equiv RECORD
         CASE : CARDINAL OF
            1: R: REAL     |
            2: C: LONGCARD
         END (*case*)
      END (* Equiv *);
BEGIN Equiv.R := 1.0/3.0;
      WriteLongCard(Equiv.C, 12);

(** example 2: POINTERS ********************)
FROM  SYSTEM IMPORT ADR;
VAR   R    : REAL;
      P    : POINTER TO LONGCARD;
```

	BEGIN R := 1.0/3.0; P := ADR(R); WriteLongCard(P ,12);
EXP	exp (lower case) import from MathLib0.
.FALSE.	FALSE
FLOAT	FLOAT
FORMAT	See *input-output* and *Z-format*
FUNCTION	PROCEDURE. A sample integer function follows: PROCEDURE TWICE(X:INTEGER):INTEGER; BEGIN RETURN 2*X (* see below *) END TWICE; The value returned is specified by the RETURN statement. The function's name must appear in the END statement.
.GE.	>=
GOTO	There is no GOTO as it's Modula philoslphy to do program control with IF-THEN-ELSE or CASE statements.
GOTO (computed)	See *case*
.GT.	>
IF	(* example 1 *) IF J<0 THEN J:=0 END; (* example 2 *) IF X<Y THEN Z:=X ELSE Z:=Y END; (* example 3 *) IF I<5 THEN I := 1; J := 2; ELSE I := 6; J := 7; END; (* example 4 *) IF code=1 THEN name="CAT" ELSIF code=2 THEN name="COW" ELSIF code=3 THEN name="DOG" ELSE name="UNKNOWN" END;
IFIX	TRUNC
IMPLICIT	Not needed: Everything must be declared.

input-output

There is no general purpose print or read statement. Instead, input-output is done with procedures, and there has to be a different procedure for every data type. Moreover these procedures can't just be used, they are in procedure libraries and must be imported.

Different implementors may supply different input-output modules. Commonly the procedures print at most one number or character per call, with no procedures provided for structured data types except one dimensional character arrays

Some selected procedures from Stony Brook Modula follow, those above the line ==== are standard and available in most other implementations.

module InOut .....

sample use	data type	format
WriteInt(I,5);	INTEGER	I5
WriteInt(I,w);	INTEGER	Iw
WriteCard(C,7);	CARDINAL	I7
WriteHex(C,4);	CARDINAL	Z4
Write(A);	CHAR	A1
WriteString(S);	ARRAY OF CHAR	A{length of S}
WriteLn;		/
ReadInt(I)	INTEGER	free format
ReadCard(C)	CARDINAL	free format
Read(C)	CHAR	A1
ReadString(C)	ARRAY OF CHAR	free format

module RealInOut .....

sample use	data type	format
WriteReal(R,10);	REAL	E10.*
WriteLongReal(R,18);	LONGREAL	E18.*
* is chosen by the procedure.		
ReadReal(R,);	REAL	free
ReadLongReal(R,18);	LONGREAL	free
========== Procedures below this line are not standard.		
WriteFixed(R,7,3);	REAL, LONGREAL	F9.4
WriteFixed(R,w-2,w-2-d);	REAL, LONGREAL	Fw.d

module LongInOut .....

sample use	data type	format
WriteLong(I,15);	LONGINT	I15
WriteLongCard(C,17);	LONGCARD	I17
ReadLong(I);	LONGINT	free format
ReadLongCard(C);	LONNGCARD	free format

FORTRAN (or Topic)	MODULA (or information)
INTEGER I, J INTEGER LIST(8) INTEGER LIST(0:7)	VAR I, J : INTEGER; VAR LIST : ARRAY [1..8] OF INTEGER; VAR LIST : ARRAY [0..7] OF INTEGER;
.LE.	<=
logarithm	ln (lower case) import from MathLib0.
LOGICAL L LOGICAL LL(16)	VAR L : BOOLEAN; VAR LL : ARRAY [1..16] OF BOOLEAN;
LOOP	LOOP (* example *) LOOP; J := J+1; IF J > 5 THEN EXIT; END (*loop*);
MAX(A, B)	Not available. Use IF A>B THEN C:=A ELSE C:=B END; (MAX is a built-in Modula function, but it has *no* relation to the fortran MAX. In Modula the argument of MAX is a simple ordered type and MAX(type) returns the largest value that type can take.)
MIN(A, B)	Not available. Use IF A<B THEN C:=A ELSE C:=B END; (MIN is a built-in Modula function, but it has *no* relation to the fortran MIN. In Modula the argument of MIN is a simple ordered type and MIN(type) returns the least value that type can take.)
MOD(X,3)	X MOD 3
.NOT.	NOT
.NE.	# or <>
.OR.	OR
PARAMETER (PI=3.141)	CONST PI=3.141; The decimal point specifiest PI as real. You need not and must not otherwise declare PI real
parentheses	( ) to control order of computation, and to to enclose a subroutine argument list, [ ] for array indexing
PRINT	See input-output
QUIT	EXIT See also LOOP
READ	See input-output

REAL X, Y, Z REAL VECTOR(0:8)	VAR X, Y, Z : REAL; VAR VECTOR : ARRAY [0..8] OF REAL; As in Fortran, real constants are identified by a decimal point.
REAL*8	Same as above but use LONGREAL instead of REAL. (See also DOUBLE PRECISION and CONSTANT.)
RETURN	RETURN; In a function: RETURN value_to_return;
SIN	sin (lower case) import from MathLib0.
SQRT	sqrt (lower case) import from MathLib0.
STOP	HALT

SUBROUTINE	PROCEDURE.
	
	A sample subroutine to clear a vector follows:
	PROCEDURE VectorClear (VAR V: ARRAY OF LONGREAL);
	VAR I: CARDINAL;
	BEGIN;
	FOR I := 0 TO HIGH(V) DO;
	V[I] := 0.0;
	END;
	END VectorClear;

TAN	tan (lower case) import from MathLib0.
THEN	THEN See also IF
.TRUE.	TRUE

while loop	WHILE logical expression DO
	(* statements *)
	END;

WRITE	See input-output
Z format	WriteHex from module InOut only writes cardinals. (For mapping bit patterns of other types to CARDINAL, see EQUIVALENCE.) HexOut from module IO, as modified at U.N.B, will write anything in hex (If your version of IO does'nt have HexOut, W. Knight has the source.)