CR-171952 c.1

TRW

47467-H005-UX-00

ORBITAL FLIGHT SIMULATION UTILITY SOFTWARE UNIT SPECIFICATIONS

(NASA-CR-171952) ORBITAL FLIGHT SIMULATION UTILITY SOFTWARE UNIT SPECIFICATIONS (TRW Systems Group) 194 p CSCL 14B

N87-13468

Unclas G3/14 43627

21 APRIL 1986

Prepared for

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

Contract NAS9-17554

System Development Division
TRW Defense Systems Group
Houston, Texas

47467-H005-UX-00

ORBITAL FLIGHT SIMULATION
UTILITY SOFTWARE UNIT
SPECIFICATIONS

21 APRIL 1986

Prepared for

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

Contract NAS9-17554

System Development Division
TRW Defense Systems Group
Houston, Texas

47467-H005-UX-00

ORBITAL FLIGHT SIMULATION

UTILITY SOFTWARE UNIT

SPECIFICATIONS

21 April 1986

Prepared for

National Aeronautics and Space Administration

Lyndon B. Johnson Space Center

Contract NAS9-17554

Prepared by S. W. Wilson

S. W. Wilson
Staff Engineer
Systems Engineering and
Analysis Department

Approved by D K Phillips

D. K. Phillips
Manager
Systems Engineering and
Analysis Department

System Development Division

TRW
Defense Systems Group
Houston, Texas

FOREWORD

The HP PASCAL source code contained in pages 6 through 104 was developed for the Mission Planning and Analysis Division (MPAD) of NASA/JSC, and takes the place of detailed flow charts defining the specifications for a Utility Software Unit designed to support orbital flight simulators such as MANHANDLE and GREAS (General Research and Engineering Analysis Simulator). Besides providing basic input/output, mathematical, vector, matrix, quaternion, and statistical routines for such simulators, one of the primary functions of the Utility Software Unit is to isolate all system-dependent code in one well-defined compartment, thereby facilitating transportation of the simulations from one computer to another.

Pages 1 and 2, respectively, contain directives to the PASCAL compilers of the HP-9000 Series 200 Pascal 3.0 operating system and the HP-9000 Series 500 HP-UX 5.0 operating system that produce --- in each system --- a single file of relocatable code from four separate files of source code. Three of the source code files ('Utilmath.I', 'Utilvemq.I', and 'Utilstat.I') are common to both operating systems. The fourth source code file ('UTILSPIF.I' for the Pascal 3.0 system and 'utilspif.I' for the HP-UX 5.0 system) contains all of the system-dependent PASCAL code for the Utility Software Unit. A fifth file of source code written in the C language ('utilscif.c', listed on pages 3 through 5) is required to interface 'utilspif.I' with the HP-UX "curses" hardware-independent terminal input/output package, which uses "# define" statements extensively to define C pseudofunctions that cannot be called directly from a PASCAL routine.

Pages 105 through 107 contain the Pascal 3.0 compiler directives and the driver source code for a unit test program. Its counterpart for the HP-UX 5.0 operating system is contained in pages 108 through 110. The major portion of the unit test program source code (pages 111 through 136) is common to both operating systems. Unit test results from the Pascal 3.0 operating system are shown on pages 137 through 152. The results from the HP-UX operating system are shown on pages 153 through 168.

Because its portability is limited to Hewlett-Packard computers, HP PASCAL is not regarded as the best language for ultimate implementation of general purpose software. However, pending the availability of an ADA compiler for MPAD's mainline computers, its unique features and versatility make it the best medium at hand for the development, testing, and maintenance of software specifications (which, as distinguished from executable code, is the primary subject of this document). The fact that interim or prototype executable programs can be implemented (on Hewlett-Packard computers) simply by compiling the source code that serves as the software specification is considered to be advantageous, but it was not the reason for choosing HP PASCAL as the medium for specification.

TABLE OF CONTENTS

* HP-9000 Model 216 / Pascal 3.0 Operating System

** HP-9000 Series 500 / HP-UX 5.0 Operating System

```
{ begin File 'UTILUNIT.TEXT' }

{ Utility Software Unit for HP-9000 Model 216 with Pascal 3.0 Op Sys }


{ NASA/JSC/MPAD/TRW      Sam Wilson }
{ Updated  Fri Apr 11 01:43:49 1986 }

{ This construct binds several related PASCAL modules into a single compila-  }
{ tion unit so that the relocatable code produced by the compiler will reside }
{ in one file rather than in many, thus simplifying input instructions to the }
{ operating system's linker/loader when combining it with other relocatable   }
{ code to produce an executable program.  To simplify editing, the source     }
{ code for each functional module resides in a separate file, each being      }
{ named in an "include" directive to the compiler (see below).                }

{ Except for the INVERT_MATRIX and DIAGONALIZE_SYMMATRIX procedures in module }
{ UTILMATH, the input parameters for every exported function and procedure    }
{ defined in the Utility Software Unit are passed "by value" rather than      }
{ "by reference".  This means that the input values are assigned to local     }
{ variables within the called routine, and that no reference to an input var- }
{ iable by the called routine can modify anything other than the local copy.  }
{ It also makes it possible in the calling routine to substitute any expres-  }
{ sion of the proper type (even a named or a literal constant) for any formal }
{ input parameter in the argument list of the called routine, thus often      }
{ simplifying the code quite a bit.  For instance, the single statement       }
{                                                                             }
{      OUTPUT_ANG := ANGDEG( ATAN2( 1.5L0, DOTP( POS, ZUNVEC ) ) ) ;          }
{                                                                             }
{ would have to be replaced by a sequence something like                      }
{                                                                             }
{      V := ZUNVEC ;                                                          }
{      X := DOTP( POS, V ) ;                                                  }
{      Y := 1.5L0 ;                                                           }
{      A := ATAN2( Y, X ) ;                                                   }
{      OUTPUT_ANG := ANGDEG( A ) ;                                            }
{                                                                             }
{ if the inputs to ANGDEG, ATAN2, and DOTP were passed by reference, because  }
{ only a variable name is allowed to replace a formal parameter that is       }
{ passed by reference.                                                        }


$ Sysprog On $
$ Switch_strpos $
{ $ list off $ }   $ include 'Utilmath.I.' $   { $ list on $ }
{ $ list off $ }   $ include 'UTILSPIF.I.' $   { $ list on $ }
{ $ list off $ }   $ include 'Utilvemq.I.' $   { $ list on $ }
{ $ list off $ }   $ include 'Utilstat.I.' $   { $ list on $ }

module UTILDUMMY ;          { no function; just terminates compilation tidily }
export type SOMETHING_TO_SATISFY_COMPILER = boolean ;
implement
end . { File 'UTILUNIT.TEXT' }
```

```
{ begin File 'utilunit.p' }

{ Utility Software Unit for HP-9000 Series 500 with HP-UX 5.0 Op Sys }


{ NASA/JSC/MPAD/TRW      Sam Wilson }
{ Updated  Thu Apr 10 23:01:06 1986 }

{ This construct binds several related PASCAL modules into a single compila-  }
{ tion unit so that the relocatable code produced by the compiler will reside }
{ in one file rather than in many, thus simplifying input instructions to the }
{ operating system's linker/loader when combining it with other relocatable   }
{ code to produce an executable program.  To simplify editing, the source     }
{ code for each functional module resides in a separate file, each being      }
{ named in an "include" directive to the compiler (see below).                }

{ Except for the INVERT_MATRIX and DIAGONALIZE_SYMMATRIX procedures in module }
{ UTILMATH, the input parameters for every exported function and procedure    }
{ defined in the Utility Software Unit are passed "by value" rather than      }
{ "by reference".  This means that the input values are assigned to local    }
{ variables within the called routine, and that no reference to an input var- }
{ iable by the called routine can modify anything other than the local copy.  }
{ It also makes it possible in the calling routine to substitute any expres-  }
{ sion of the proper type (even a named or a literal constant) for any formal }
{ input parameter in the argument list of the called routine, thus often      }
{ simplifying the code quite a bit.  For instance, the single statement       }
{                                                                             }
{      OUTPUT_ANG := ANGDEG( ATAN2( 1.5L0, DOTP( POS, ZUNVEC ) ) ) ;          }
{                                                                             }
{ would have to be replaced by a sequence something like                      }
{                                                                             }
{      V := ZUNVEC ;                                                          }
{      X := DOTP( POS, V ) ;                                                  }
{      Y := 1.5L0 ;                                                           }
{      A := ATAN2( Y, X ) ;                                                   }
{      OUTPUT_ANG := ANGDEG( A ) ;                                            }
{                                                                             }
{ if the inputs to ANGDEG, ATAN2, and DOTP were passed by reference, because  }
{ only a variable name is allowed to replace a formal parameter that is       }
{ passed by reference.                                                        }

$ standard_level 'hp_modcal' $
$ type_coercion 'noncompatible' $
{ $ list off $ }   $ include 'Utilmath.I' $   { $ list on $ }
$ type_coercion 'conversion' $
{ $ list off $ }   $ include 'utilspif.I' $   { $ list on $ }
{ $ list off $ }   $ include 'Utilvemq.I' $   { $ list on $ }
{ $ list off $ }   $ include 'Utilstat.I' $   { $ list on $ }

module UTILDUMMY ;          { no function; just terminates compilation tidily }
export type SOMETHING_TO_SATISFY_COMPILER = boolean ;
implement
end . { File 'utilunit.p' }
```

```
/*  begin File 'utilscif.c'  */

/*  Utility Software Unit for HP-9000 Series 500 with HP-UX 5.0 Op Sys  */

void main() {}    /*  Subject  :  System/C Interface  */
                  /*  Domain   :  Universal           */

                  /*  NASA/JSC/TRW          Sam Wilson  */
                  /*  Updated  Thu Apr 10 23:13:32 1986  */

                  /*  This compilation unit contains the C functions  */
                  /*  required to interface the PASCAL code with the  */
                  /*  appropriate HP-UX utility routines.             */

# include <curses.h>

/*------------------------------------------------------------------------ */

void clearline()
     {
     int row ;
     int col ;
     getyx( stdscr, row, col ) ;
     move( row, 0 ) ;
     clrtoeol() ;
     refresh() ;
     }

/*------------------------------------------------------------------------ */

void clearscreen()
     {
     clear() ;
     refresh() ;
     }

/*------------------------------------------------------------------------ */

void fetchpac( s ) char *s ;
     {
     getstr( s ) ;
     }

/*------------------------------------------------------------------------ */

void ioinitialize()
     {
     initscr() ;
     refresh() ;
     }

/*------------------------------------------------------------------------ */

void iouninitialize ()
     {
     endwin() ;
     exit( 0 ) ;
     }

/*------------------------------------------------------------------------ */
```

```
void moveup1row()
     {
     int row ;
     int col ;
     getyx( stdscr, row, col ) ;
     move( row-1, col ) ;
     refresh() ;
     }

/*------------------------------------------------------------------------- */

void setbrightblinki()
     {
     attrset( A_BLINK ) ;
     }

/*------------------------------------------------------------------------- */

void setcbreak()
     {
     cbreak() ;
     }

/*------------------------------------------------------------------------- */

void setcecho()
     {
     echo() ;
     }

/*------------------------------------------------------------------------- */

void setcwait()
     {
     nodelay( stdscr, FALSE ) ;
     }

/*------------------------------------------------------------------------- */

void sethalfbrighti()
     {
     attrset( A_DIM ) ;
     }

/*------------------------------------------------------------------------- */

void setnocbreak()
     {
     nocbreak() ;
     }

/*------------------------------------------------------------------------- */

void setnocecho()
     {
     noecho() ;
     }

/*------------------------------------------------------------------------- */
```

```
void setnocwait()
     {
     nodelay( stdscr, TRUE ) ;
     }

/*------------------------------------------------------------------------ */

void setnormalvideo()
     {
     attrset( A_NORMAL ) ;
     }

/*------------------------------------------------------------------------ */

void showpac( s ) char *s ;
     {
     addstr( s ) ;
     refresh() ;
     }

/*------------------------------------------------------------------------ */

void soundalert()
     {
     beep() ;
     }

/*------------------------------------------------------------------------ */

int kbdcharcode()
     {
     return getch() ;
     }

/*------------------------------------------------------------------------ */

/*  end File 'utilscif.c'  */
```

```
$ page $  { begin File 'Utilmath.I' }

{ Utility Software Unit for HP-9000 Series 200/300/500 Computers }

module UTILMATH ;  { Subject  :  Mathematics }
                   { Domain   :  Universal   }

                   { NASA/JSC/MPAD/TRW      Sam Wilson }
                   { Updated  Sat Apr 12 15:23:58 1986 }

export

     const

          UNITOL =  4.0L-14 ;      { arithmetical error tolerance for computed }
                                   { longreal values on the order of unity     }

          ZERO  =   0.0L0 ;
          ONE   =   1.0L0 ;
          TWO   =   2.0L0 ;
          THREE =   3.0L0 ;
          FOUR  =   4.0L0 ;
          FIVE  =   5.0L0 ;
          SIX   =   6.0L0 ;
          SEVEN =   7.0L0 ;
          EIGHT =   8.0L0 ;
          NINE  =   9.0L0 ;
          TEN   =  10.0L0 ;

          DEGPERRAD =  5.7295779513082325L+1 ;          { degrees per radian }
          HAFPI     =  1.5707963267948966L+0 ;                      { pi / 2 }
          PI        =  3.1415926535897932L+0 ;                          { pi }
          RADPERDEG =  1.7453292519943295L-2 ;          { radians per degree }
          TWOPI     =  6.2831853071795864L+0 ;                      { 2 * pi }

          MAXMATORDER     = 12 ; { max order of matrx to be invrtd or diagnlzd }
          MAXSQUAREINDEX = MAXMATORDER * MAXMATORDER ;
          MAXTRIANGINDEX = ( MAXMATORDER * ( MAXMATORDER + 1 ) ) div 2 ;
```

```
$ page $

     type

          MATROWCOLNUM = 1..MAXMATORDER ;

          DIAGMAT  =                                   { an array in which     }
               array [ 1..MAXMATORDER ] of longreal ;  { the nonzero elements  }
                                                       { of a diagonal         }
                                                       { matrix M of order     }
                                { n <= MAXMATORDER are stored in the sequence }
                                {                                              }
                                {      M[1,1], M[2,2], ......, M[n,n].         }
                                {                                              }


          TRIANGMAT =                                   { an array in which    }
               array [ 1..MAXTRIANGINDEX ] of longreal; { the nonzero elemnts  }
                                                        { of a triangular      }
                                                        { matrix M of order    }
                                { n <= MAXMATORDER are stored in the sequence }
                                {                                              }
                                {      M[1,1],                                 }
                                {      M[1,2], M[2,2],                         }
                                {      ......, ......, ......,                 }
                                {      M[1,n], M[2,n], ......, M[n,n]          }
                                {                                              }
                                { or                                           }
                                {                                              }
                                {      M[1,1],                                 }
                                {      M[2,1], M[2,2],                         }
                                {      ......, ......, ......,                 }
                                {      M[n,1], M[n,2], ......, M[n,n],         }
                                {                                              }
                                { depending on whether M is an upper or a      }
                                { lower triangular matrix.                     }


          SQUAREMAT =                                    { an array in which   }
               array [ 1..MAXSQUAREINDEX ] of longreal ; { the elements        }
                                                         { of a square         }
                                                         { matrix M of order   }
                                { n <= MAXMATORDER are stored in the sequence }
                                {                                              }
                                {      M[1,1], M[1,2], ......, M[1,n],         }
                                {      M[2,1], M[2,2], ......, M[2,n],         }
                                {      ......, ......, ......, ......,         }
                                {      M[n,1], M[n,2], ......, M[n,n],         }
                                {                                              }
                                { which is the PASCAL storage order for the    }
                                { elements of a two-dimensional n x n array.   }
```

```
$ page $

      function  INT( X : longreal ) : integer ;

                { INT is equivalent to the HPL "int" function (also known as   }
                { the "floor" function).  Its value, which is returned to the  }
                { calling routine on the stack, is the greatest integer <= X.  }
                { NOTE:  INT( X ) is not equal to trunc( X ) when  X < 0 .     }


      function  FRAC( X : longreal ) : longreal ;

                { FRAC is equivalent to the HPL "frc" function, which is never }
                { negative.  Its value, X - INT( X ), may be different from    }
                { the casual expectation when X < 0 .                          }


      function  RMOD( Y, X : longreal ) : longreal ;

                { The value of RMOD( Y, X ) is zero when X = 0; otherwise its  }
                { value is Y - X * INT( Y/X ).  RMOD( Y, X ) is in a sense the }
                { (long)real equivalent of the integer expression "J mod I".   }


      function  RSIGN( X : longreal ) : integer ;

                { This is the "sign" function of a (long)real number.  Its     }
                { value is -1 if X < 0 ; otherwise its value is +1 .           }


      function  ISIGN( I : integer  ) : integer ;

                { This is the "sign" function of an integer number.  Its       }
                { value is -1 if X < 0 ; otherwise its value is +1 .           }


      function  IMAX( J, I : integer  ) : integer ;

                { This is the integer "maximum" function.  Its value is the    }
                { greater of its two arguments.                                }


      function  IMIN( J, I : integer  ) : integer ;

                { This is the integer "minimum" function.  Its value is the    }
                { lesser of its two arguments.                                 }


      function  RMAX( Y, X : longreal ) : longreal ;

                { This is the (long)real "maximum" function.  Its value is the }
                { greater of its two arguments.                                }


      function  RMIN( Y, X : longreal ) : longreal ;

                { This is the (long)real "minimum" function.  Its value is the }
                { lesser of its two arguments.                                 }
```

```
$ page $

     function  ANGDEG( X : longreal ) : longreal ;

               { This function converts an angle from radian measure to de-    }
               { grees.  Normally it is used to convert internal values to     }
               { measurement units suitable for output.  ANGDEG is unique in   }
               { that angles output by every other function defined in this    }
               { module are measured in radians.                               }


     function  ANGRAD( X : longreal ) : longreal ;

               { This function converts an angle from degree measure to rad-   }
               { ians.  Normally it is used to convert external values sup-    }
               { plied by the user to measurement units suitable for internal  }
               { computations.  ANGRAD is unique in that the input value must  }
               { of course be measured in degrees, whereas angles input to     }
               { every other function defined in this module must be measured  }
               { in radians.                                                   }


     function  ANG1( X : longreal ) : longreal ;

               { The value of ANG1( X ) is the angular equivalent of X that    }
               { lies in the range of  0 <= ANG1(X) < TWOPI.                   }


     function  ANG2( X : longreal ) : longreal ;

               { The value of ANG2( X ) is the angular equivalent of X that    }
               { lies in the range of  -PI < ANG2(X) <= PI.                    }


     function  ATAN2( Y, X : longreal ) : longreal ;

               { The value returned by this function subprogram always lies    }
               { in the range of  -PI < ATAN2(Y,X) <= PI  and is equal to      }
               { arctangent( Y/X ) in the quadrant where the sine and cosine   }
               { of the angle have the signs of Y and X, respectively.  It is  }
               { equivalent to the FORTRAN function of the same name, except   }
               { that ATAN2(0,0) --- undefined in FORTRAN --- is zero.         }

     function  ATAN1( Y, X : longreal ) : longreal ;

               { ATAN1 is similar to ATAN2, except  0 <= ATAN1(Y,X) < TWOPI.   }
```

```
$ page $

      function  HMS( X : longreal ) : longreal ;

                { HMS converts X (time measured in seconds) to hours, minutes, }
                { and seconds, and returns the result packed into a longreal   }
                { number.  HMS( X ) has the sign of X and the form hmm.ssf,    }
                { where h represents however many decimal digits are required  }
                { to express the number of whole hours, mm and ss represent    }
                { the number of additional whole minutes and seconds, and f    }
                { represents the remaining decimal fraction of a second.  For  }
                { example, HMS( 36385.874 ) = 1006.25874 .                     }


      function  SECS( X : longreal ) : longreal ;

                { SECS is the inverse of the HMS function.  It converts a time }
                { X --- expressed in hours, minutes, and seconds, and packed   }
                { into a longreal number of the form hmm.ssf --- to the equiv- }
                { alent number of seconds, returning the result as a longreal  }
                { number.  For example, SECS( 1006.25874 ) = 36385.874 .       }


      function  JULIAN_DAYNUM( YEAR , MONTH , DAY : integer ) : integer ;

                { This function returns the number of the Julian day beginning }
                { at noon on the date defined by YEAR, MONTH, and DAY numbers  }
                { in the Gregorian (i.e., civil) calendar.  For example,       }
                { JULIAN_DAYNUM( 1980, 4, 2 ) = 2444332 is the number of the   }
                { Julian day beginning at noon on 2 April 1980.                }


      function  TRIANG_INDEX( i, j : MATROWCOLNUM ) : integer ;

                { The value of this function is the one-dimensional index (in  }
                { an array of the type TRIANGMAT) of the element M[i,j] from a }
                { triangular matrix M.                                         }
```

```
$ page $


     procedure INVERT_MATRIX ( anyvar MATRIX  : SQUAREMAT     ;
                                      ORDER   : MATROWCOLNUM ;
                               anyvar INVERSE : SQUAREMAT     ) ;

          { Given (in MATRIX) the elements of a square matrix of order   }
          { no greater than MAXMATORDER, this procedure will compute the }
          { elements of its inverse (if it exists) and return them to    }
          { the calling routine in the output variable INVERSE.  The     }
          { "anyvar" notation in the parameter list causes the normal    }
          { PASCAL type-checking rules to be relaxed, and makes it per-  }
          { missable in the calling routine to substitute, for MATRIX    }
          { and INVERSE, the names of variables that are not actually    }
          { declared to be of the type SQUAREMAT.  For example, the      }
          { calling routine might legitimately contain the following     }
          { statements:                                                  }
          {                                                              }
          {      type                                                    }
          {                                                              }
          {           MAT4X4 = array [ 1..4, 1..4 ] of longreal ;        }
          {           MAT9X9 = array [ 1..9, 1..9 ] of longreal ;        }
          {                                                              }
          {      var                                                     }
          {                                                              }
          {           L : MAT4X4 ;                                       }
          {           M : MAT4X4 ;                                       }
          {           N : MAT9X9 ;                                       }
          {                                                              }
          {      begin                                                   }
          {      .                                                       }
          {      INVERT_MATRIX ( L, 4, M ) ;                             }
          {      INVERT_MATRIX ( N, 9, N ) ;                             }
          {      .                                                       }
          {      end ;                                                   }
          {                                                              }
          { As indicated by the second reference to INVERT_MATRIX, it    }
          { is permissable to substitute the same variable name for      }
          { MATRIX and INVERSE if one wishes to overwrite the original   }
          { matrix with its inverse.                                     }

          { Program execution will be aborted with an escapecode of 9901 }
          { if the input matrix turns out to be singular, and no value   }
          { will be assigned to INVERSE.  The reference to INVERT_MATRIX }
          { should be embedded in a "try/recover" construct if it is     }
          { desired to provide exception-handling code in the calling    }
          { routine to recover from such an eventuality.                 }
```

```
$ page $

   procedure DIAGONALIZE_SYMMATRIX ( anyvar SYMMET   : TRIANGMAT    ;
                                            ORDER    : MATROWCOLNUM ;
                                            TOLRATIO : longreal     ;
                                     anyvar DIAG     : DIAGMAT      ;
                                     anyvar ORTHOG   : SQUAREMAT    ) ;

        { Given (in SYMMET) the unique elements of a symmetric matrix   }
        { S of order no greater than MAXMATORDER, this procedure uses   }
        { the Jacobi method of iteration to find an orthogonal matrix   }
        { M that will transform an unknown diagonal matrix D into S by }
        { use of the equation S = T * D * M, where "*" is the matrix    }
        { multiplication operator and T is the transpose of M.  Itera- }
        { tion ceases when a value of M is found such that every        }
        { off-diagonal element of an approximated diagonal matrix,      }
        { D' = M * S * T, has an absolute value no greater the product }
        { of the input parameter TOLRATIO with the root-mean-square     }
        { magnitude of the diagonal elements of D (which can be com-    }
        { puted easily even though the individual elements of D are     }
        { unknown).  After the convergence test is satisfied, the di-   }
        { agonal elements of the approximation D' are returned to the   }
        { calling routine as components of the output variable DIAG     }
        { and the final value of M is returned in the output variable   }
        { ORTHOG.

        { If D' has not converged to the specified tolerance after a    }
        { reasonable number of iterations (50 times the order of S),    }
        { the values of M and D' are NOT returned to the calling        }
        { routine, and program execution is aborted with an escapecode }
        { of 9902.  The reference to DIAGONALIZE_SYMMATRIX should be    }
        { embedded in a "try/recover" construct if it is desired to     }
        { provide exception-handling code in the calling routine to     }
        { recover from such an eventuality.                             }

        { As in the case of the INVERT_MATRIX procedure, use of the     }
        { "anyvar" notation in the parameter list makes it possible in }
        { the calling routine to substitute (for SYMMET, DIAG, and      }
        { ORTHOG) the names of variables that are not actually de-      }
        { clared to be of the types indicated in the formal parameter   }
        { list.  The DIAGONALIZE procedure code in the UTILVEMQ module }
        { contains an example of DIAGONALIZE_SYMMATRIX usage.           }
```

```
$ page $

implement

     type

          DIAGPOINTER    = ^DIAGMAT   ;
          SQUAREPOINTER = ^SQUAREMAT ;
          TRIANGPOINTER = ^TRIANGMAT ;

     var
          INV      : SQUAREPOINTER ;
          MAT      : SQUAREPOINTER ;
          MATORDER : integer       ;
          SYM      : TRIANGPOINTER ;

     function  SQUARE_INDEX  ( i, j : MATROWCOLNUM ) : integer ; forward ;
     procedure EXCHANGE_ROWS ( i, j : MATROWCOLNUM )           ; forward ;
```

```
$ page $


function INT( X : longreal ) : integer;

     var

          I : integer ;

     begin
     I := trunc( X ) ;
     if X < ZERO then
          if X <> I then
               I := I - 1 ;
     INT := I ;
     end ;


function FRAC( X : longreal ) : longreal ;

     var

          F : longreal ;

     begin
     F := X - trunc( X ) ;
     if F < ZERO then
          F := F + ONE ;
     FRAC := F ;
     end ;


function RMOD( Y , X : longreal ) : longreal ;

     begin
     if X = ZERO
          then RMOD := ZERO
          else RMOD := Y - X * INT( Y / X ) ;
     end ;
```

```
$ page $

function RSIGN( X : longreal ) : integer ;

     begin
     if X < ZERO
          then RSIGN := -1
          else RSIGN :=  1 ;
     end ;


function ISIGN( I : integer ) : integer ;

     begin
     if I < 0
          then ISIGN := -1
          else ISIGN :=  1 ;
     end ;


function IMAX( J , I : integer ) : integer ;

     begin
     if I > J
          then IMAX := I
          else IMAX := J ;
     end ;


function IMIN( J , I : integer ) : integer ;

     begin
     if I < J
          then IMIN := I
          else IMIN := J ;
     end ;


function RMAX( Y , X : longreal ) : longreal ;

     begin
     if Y > X
          then RMAX := Y
          else RMAX := X ;
     end ;


function RMIN( Y , X : longreal ) : longreal ;

     begin
     if Y < X
          then RMIN := Y
          else RMIN := X ;
     end ;
```

```
$ page $


function ANGDEG( X : longreal ) : longreal ;

     begin
     ANGDEG := X * DEGPERRAD ;
     end ;


function ANGRAD( X : longreal ) : longreal ;

     begin
     ANGRAD := X * RADPERDEG ;
     end ;


function ANG1( X : longreal ) : longreal ;

     begin
     ANG1 := TWOPI * FRAC( X / TWOPI ) ;
     end ;


function ANG2( X : longreal ) : longreal ;

     var

          A : longreal ;

     begin
     A := TWOPI * FRAC( X / TWOPI ) ;
     if A > PI then  A := A - TWOPI ;
     ANG2 := A ;
     end ;
```

```
$ page $


function ATAN2( Y , X : longreal ) : longreal ;

     var

          A   : longreal ;
          XSQ : longreal ;
          YSQ : longreal ;

     begin
     YSQ := sqr( Y ) ;
     XSQ := sqr( X ) ;
     if( YSQ + XSQ ) = ZERO

          then A := ZERO

          else
          begin
          if YSQ > XSQ

               then
               A := HAFPI - arctan( X / abs( Y ) )

               else
               begin
               A := arctan( abs( Y / X ) ) ;
               if X < ZERO then A := PI - A ;
               end ;

          if Y < ZERO then A := -A ;
          end ;

     ATAN2 := A ;
     end ;


function ATAN1( Y , X : longreal ) : longreal ;

     begin
     ATAN1 := ANG1( ATAN2( Y, X ) ) ;
     end ;
```

```
$ page $


function HMS( X : longreal ) : longreal ;

     var

          A : longreal ;
          H : integer  ;
          M : integer  ;
          S : longreal ;

     begin
     A := abs( X ) ;
     H := trunc( A/3600 ) ;
     M := trunc( A/60 ) - 60*H ;
     S := A - 60 * ( M + 60*H ) ;
     HMS := RSIGN( X ) * ( 100*H + M + S/100 ) ;
     end ;


function SECS( X : longreal ) : longreal ;

     var

          A : longreal ;
          H : integer  ;
          M : integer  ;
          S : longreal ;

     begin
     A := abs( X ) ;
     H := trunc( A/100 ) ;
     M := trunc( A ) - 100*H ;
     S := 100 * ( A - M - 100*H ) ;
     SECS := RSIGN( X ) * ( 60 * ( 60*H + M ) + S ) ;
     end ;


function JULIAN_DAYNUM( YEAR , MONTH , DAY : integer ) : integer ;

     var

          D : integer ;
          I : integer ;

     begin
     if MONTH < 3
          then I := 1
          else I := 0 ;
     D := DAY - 32075 + ( 1461 * ( YEAR + 4800 - I ) ) div 4 ;
     D := D + ( 367 * ( MONTH - 2 + 12*I ) ) div 12 ;
     D := D - ( 3 * ( ( YEAR + 4900 - I ) div 100 ) ) div 4 ;
     JULIAN_DAYNUM := D ;
     end ;
```

```
$ page $


function  TRIANG_INDEX( i, j : MATROWCOLNUM ) : integer ;

     var

          h : MATROWCOLNUM ;
          k : MATROWCOLNUM ;

     begin
     h := IMIN( i, j ) ;
     k := IMAX( i, j ) ;
     TRIANG_INDEX := h + ( k * ( k- 1 ) ) div 2 ;
     end ;


function  SQUARE_INDEX( i, j : MATROWCOLNUM ) : integer ;

     begin
     SQUARE_INDEX := j + ( i - 1 ) * MATORDER ;
     end ;
```

```
$ page $


procedure INVERT_MATRIX ( anyvar MATRIX   : SQUAREMAT    ;
                                 ORDER    : MATROWCOLNUM ;
                          anyvar INVERSE  : SQUAREMAT    ) ;

     var

          BESTROW : MATROWCOLNUM ;
          BESTVAL : longreal     ;
          i       : MATROWCOLNUM ;
          ij      : integer      ;
          ik      : integer      ;
          j       : MATROWCOLNUM ;
          k       : MATROWCOLNUM ;
          kj      : integer      ;
          kk      : integer      ;
          MIK     : longreal     ;
          X       : longreal     ;
          Y       : longreal     ;
          Z       : longreal     ;

     begin
     MATORDER := ORDER ;
     new ( MAT ) ;              { allocate storage for working copy of MATRIX  }
     new ( INV ) ;              { allocate storage for working copy of INVERSE }
     BESTVAL := ZERO ;
     for i := 1 to MATORDER do
          for j := 1 to MATORDER do
               begin
               ij := SQUARE_INDEX( i, j ) ;
               if j = i
                    then INV^[ij] := ONE
                    else INV^[ij] := ZERO ;
               MAT^[ij] := MATRIX[ij] ;
               if j = 1 then
                    begin
                    X := abs( MAT^[ij] ) ;
                    if X > BESTVAL then
                         begin
                         BESTVAL := X ;
                         BESTROW := i ;
                         end ;
                    end ;
               end ;
```

```
$ page $

          for k := 1 to MATORDER do
               begin { k loop }
               if BESTVAL = ZERO then escape ( 9901 ) ;          { MATRIX is singular }
               if BESTROW <> k then
                    EXCHANGE_ROWS( k, BESTROW ) ;
               kk := SQUARE_INDEX( k, k ) ;
               X := ONE / MAT^[kk] ;
               for j := 1 to MATORDER do
                    begin
                    kj := SQUARE_INDEX( k, j ) ;
                    MAT^[kj] := X * MAT^[kj] ;
                    INV^[kj] := X * INV^[kj] ;
                    end ;
               BESTVAL := ZERO ;
               for i := 1 to MATORDER do
                    if i <> k then                { nullify column k of row i in MAT }
                         begin
                         ik := SQUARE_INDEX( i, k ) ;
                         MIK := MAT^[ik] ;
                         for j := 1 to MATORDER do
                              begin
                              ij := SQUARE_INDEX( i, j ) ;
                              kj := SQUARE_INDEX( k, j ) ;
                              Y := MAT^[ij] ;
                              Z := MIK * MAT^[kj] ;
                              MAT^[ij] := Y - Z ;
                              INV^[ij] := INV^[ij] - MIK * INV^[kj] ;
                              if ( j = k+1 ) and ( i > k ) then
                                   begin
                                   X := RMAX( abs( Y ), abs( Z ) ) ;
                                   if X > ZERO then
                                        begin
                                        X := abs( MAT^[ij] / X ) ;
                                        if X > BESTVAL then
                                             begin
                                             BESTVAL := X ;
                                             BESTROW := i ;
                                             end ;
                                        end ;
                                   end ;
                              end ;
                         end ;
               end ; { k loop }
          for i := 1 to MATORDER do
               for j := 1 to MATORDER do
                    begin
                    ij := SQUARE_INDEX( i, j ) ;
                    INVERSE[ij] := INV^[ij] ;
                    end ;
          dispose( MAT ) ;                                 { release temporary storage }
          dispose( INV ) ;                                 { release temporary storage }
          end ;
```

```
$ page $


procedure EXCHANGE_ROWS ( i, j : MATROWCOLNUM ) ;

     var

          ik : integer      ;
          jk : integer      ;
          k  : MATROWCOLNUM ;
          X  : longreal     ;
          Y  : longreal     ;

     begin
     for k := 1 to MATORDER do
          begin
          ik := SQUARE_INDEX( i, k ) ;
          jk := SQUARE_INDEX( j, k ) ;
          X := MAT^[ik] ;
          Y := INV^[ik] ;
          MAT^[ik] := MAT^[jk] ;
          INV^[ik] := INV^[jk] ;
          MAT^[jk] := X ;
          INV^[jk] := Y ;
          end ;
     end ;
```

```
$ page $


procedure DIAGONALIZE_SYMMATRIX ( anyvar SYMMET   : TRIANGMAT    ;
                                         ORDER    : MATROWCOLNUM ;
                                         TOLRATIO : longreal     ;
                                  anyvar DIAG     : DIAGMAT      ;
                                  anyvar ORTHOG   : SQUAREMAT    ) ;

     var

          ADJUSTED      : boolean      ;
          CA            : longreal     ;
          CASQ          : longreal     ;
          i             : MATROWCOLNUM ;
          ii            : integer      ;
          ij            : integer      ;
          ik            : integer      ;
          ITNUM         : integer      ;
          j             : MATROWCOLNUM ;
          jj            : integer      ;
          jk            : integer      ;
          k             : MATROWCOLNUM ;
          MAXITERATIONS : integer      ;
          RMSEIGENVAL   : longreal     ;
          SA            : longreal     ;
          SASQ          : longreal     ;
          THRESH        : longreal     ;
          TOL           : longreal     ;
          X             : longreal     ;
          Y             : longreal     ;
          YSACA         : longreal     ;
          Z             : longreal     ;

     begin { procedure DIAGONALIZE_SYMMATRIX }
     MATORDER := ORDER ;
     new( SYM ) ;              { allocate storage for working copy of SYMMET }
     new( MAT ) ;              { allocate storage for working copy of ORTHOG }
     Y := ZERO ;
     Z := ZERO ;
     for i := 1 to MATORDER do
          for j := 1 to MATORDER do
               begin
               ij := SQUARE_INDEX( i, j ) ;
               if j = i
                    then MAT^[ij] := ONE
                    else MAT^[ij] := ZERO ;
               if j >= i then
                    begin
                    ij := TRIANG_INDEX( i, j ) ;
                    X := SYMMET[ij] ;
                    SYM^[ij] := X ;
                    if j = i
                         then Y := Y + sqr( X )
                         else Z := Z + sqr( X ) ;
                    end ;
               end ;
     RMSEIGENVAL := sqrt( ( Y + Z + Z ) / MATORDER ) ;
     TOL := RMSEIGENVAL * TOLRATIO ;
     THRESH := sqrt( Z ) / MATORDER ;
```

```
    MAXITERATIONS := 50 * MATORDER ;
    ITNUM := 0 ;
    repeat
         ADJUSTED := false ;
         for j := 2 to MATORDER do
              begin { j loop }
              jj := TRIANG_INDEX( j, j ) ;
              for i := 1 to j - 1 do
                   begin { i loop }
                   ii := TRIANG_INDEX( i, i ) ;
                   ij := TRIANG_INDEX( i, j ) ;
                   if abs( SYM^[ij] ) > THRESH then
                        begin { matrix adjustment }
                        X := SYM^[ii] - SYM^[jj] ;
                        Y := TWO * SYM^[ij] ;
                        Z := Y / sqrt( sqr( X ) + sqr( Y ) ) ;
                        if X < ZERO then Z := -Z ;
                        SA := Z / sqrt( TWO*(ONE+sqrt( ONE-sqr( Z ) )) ) ;
                        SASQ := sqr( SA ) ;
                        CASQ := ONE - SASQ ;
                        CA := sqrt( CASQ ) ; { CA = cosine of rotation angle }
                        YSACA := Y * SA * CA ;
                        Z := SYM^[ii] ;
                        SYM^[ii] := Z * CASQ + YSACA + SYM^[jj] * SASQ ;
                        SYM^[jj] := Z * SASQ - YSACA + SYM^[jj] * CASQ ;
                        SYM^[ij] := ZERO ; { but next rotation may change it }
                        for k := 1 to MATORDER do
                             begin
                             if k <> i then
                                  if k <> j then
                                       begin
                                       ik := TRIANG_INDEX( i, k ) ;
                                       jk := TRIANG_INDEX( j, k ) ;
                                       Z  := SYM^[ik] ;
                                       SYM^[ik] := SYM^[jk] * SA + Z * CA ;
                                       SYM^[jk] := SYM^[jk] * CA - Z * SA ;
                                       end ;
                             ik := SQUARE_INDEX( i, k ) ;
                             jk := SQUARE_INDEX( j, k ) ;
                             Z  := MAT^[ik] ;
                             MAT^[ik] := MAT^[jk] * SA + Z * CA ;
                             MAT^[jk] := MAT^[jk] * CA - Z * SA ;
                             end ;
                        ADJUSTED := true ;
                        end ; { matrix adjustment }
                   end ; { i loop }
              end ; { j loop }
         if not ADJUSTED then
              if THRESH > TOL then
                   begin { threshhold adjustment }
                   THRESH := THRESH / MATORDER ;
                   ADJUSTED := true ;
                   end ; { threshhold adjustment }
         ITNUM := ITNUM + 1 ;
         if ITNUM > MAXITERATIONS then escape ( 9902 ) ; { abort pgm exect'n }
         until not ADJUSTED ;
```

```
$ page $

     for i:= 1 to MATORDER do
          begin
          ii := TRIANG_INDEX( i, i ) ;
          DIAG[i] := SYM^[ii] ;
          for j := 1 to MATORDER do
               begin
               ij := SQUARE_INDEX( i, j ) ;
               ORTHOG[ij] := MAT^[ij] ;
               end ;
          end ;
     dispose( SYM ) ;                               { release temporary storage }
     dispose( MAT ) ;                               { release temporary storage }
     end ; { procedure DIAGONALIZE_SYMMATRIX }

end ;  { module UTILMATH & File 'Utilmath.I' }
```

```
$ page $  { begin File 'UTILSPIF.I' }

{ Utility Software Unit for HP-9000 Model 216 with Pascal 3.0 Op Sys }

module UTILSPIF ;   { Subject  :  System/PASCAL Interface }
                    { Domain   :  Universal               }

                    { NASA/JSC/MPAD/TRW      Sam Wilson }
                    { Updated  Fri Apr 11 01:44:41 1986 }

                    { This one module contains all of the system-dependent      }
                    { data and PASCAL code needed for most applications.        }

import

     Sysglobals ,   { Pascal 3.0 system module }
     Sysdevs    ,   { Pascal 3.0 system module }
     Rnd        ,   { Pascal 3.0 system module }
     General_1  ,   { Pascal 3.0 system module }
     UTILMATH   ;

export

     const

          TICKSPERSEC =  100     ;               { Series 200 clock resolution }

          DATELEN   = 24 ; { number of characters in date-and-time string      }
          LINELEN   = 80 ; { max characters in input/output line string        }
          NAMELEN   =  8 ; { max characters in "name" string                   }
          PROMPTLEN = 60 ; { max characters in prompt string for user input    }
          WORDLEN   = 20 ; { max characters in "word" string                   }
```

```
$ page $

     type

          RANDOMINT = 1..maxint-1 ;     { pseudorandom integer : 1..2147483646 }

          DATESTR    = string [ DATELEN    ] ;
          LINESTR    = string [ LINELEN    ] ;
          NAMEPAC    = packed array [ 1..NAMELEN ]  of char ;
          NAMESTR    = string [ NAMELEN    ] ;
          PROMPTSTR = string [ PROMPTLEN ] ;
          WORDSTR    = string [ WORDLEN    ] ;

          CHWAITMODE = (      { describes wait mode for CHAR_INPUT function      }
               CHWAIT,        { wait if input buffer is empty                    }
               NOCHWAIT ) ;   { don't wait if input buffer is empty              }

          CHECHOMODE = (      { describes echo mode for CHAR_INPUT function      }
               CHECHO,        { echo characters back to user terminal screen     }
               NOCHECHO ) ;   { don't echo characters back to user terminal      }

          GOTWHAT = (         { describes result of executing CHAR_INPUT functn }
               NOTHING,       { the input buffer was empty (& didn't wait)       }
               ENDOFLINE,     { got an end-of-line char (<RETURN> keystroke)     }
               SOMETHING );   { got a character that was not end-of-line         }

          CHINPUTREC =
               record  { returned by the CHAR_INPUT function }
               Q : GOTWHAT ;         { qualifier for the returned character      }
               C : char    ;         { C = ' ' if Q = NOTHING or ENDOFLINE;      }
               end ; { record }      { else C = char produced by user keystroke }


     var

          LP       : text     ; { standard file for printed output; must be      }
                                { opened by "rewrite" statement somewhere in      }
                                { program before use                              }
```

```
$ page $

      procedure INITIALIZE_IO ;

                { This procedure performs the I/O system initialization re-    }
                { quired to support most applications.  Normally it should be  }
                { the first procedure called by any program.                   }


      procedure CLEAN_UP_IO ;

                { This procedure closes any open files and generally cleans    }
                { up the I/O system.  Normally it should be the last procedure }
                { called by any program.                                       }


      procedure CLEAR_LINE ;

                { This procedure clears the line on the user terminal where    }
                { the cursor is currently located.                             }


      procedure CLEAR_SCREEN ;

                { This procedure clears the user terminal screen.              }


      procedure FETCHLN ( var STR : LINESTR ) ;

                { This procedure reads (and thus removes) characters from the  }
                { user terminal input buffer --- storing them in the string    }
                { variable STR --- until an end-of-line character (generated   }
                { by a <RETURN>* keystroke) has been read.  The end-of-line    }
                { character is always discarded, along with any other charac-  }
                { ters in excess of the number (LINELEN) needed to fill the    }
                { variable STR to its maximum capacity.                        }

                { * NOTE: Some keyboards, instead of having a <RETURN> key,    }
                {         have an <ENTER> key that serves the same purpose.    }
                {         Wherever "<RETURN>" appears in this and following    }
                {         descriptions, it should be interpreted as "<RETURN>  }
                {         or <ENTER>, whichever is present on the keyboard".   }


      procedure LOITER ( MILLISECS : integer ) ;

                { This procedure loops through meaningless code (effectively   }
                { suspending program execution) for the number of milliseconds }
                { specified in its argument.                                   }
```

```
$ page $


     procedure MOVE_UP ;

                { This procedure moves the cursor up one line on the user       }
                { terminal.                                                     }


     procedure SET_BRIGHTBLINK_INVERSE_VIDEO ;

                { This procedure causes characters to be written on the user    }
                { terminal screen in the blinking bright inverse video mode,    }
                { which remains in effect until another mode is specified.      }


     procedure SET_HALFBRIGHT_INVERSE_VIDEO ;

                { This procedure causes characters to be written on the user    }
                { terminal screen in the halfbright inverse video mode,         }
                { which remains in effect until another mode is specified.      }


     procedure SET_NORMAL_VIDEO ;

                { This procedure causes characters to be written on the user    }
                { terminal screen in the normal video mode, which remains in    }
                { effect until another mode is specified.                       }


     procedure SHOW ( STR : LINESTR ) ;

                { This procedure writes a PASCAL string to the user terminal.   }


     procedure SHOWLN ( STR : LINESTR ) ;

                { This procedure writes a PASCAL string to the user terminal,   }
                { and then positions the cursor at the beginning of the next    }
                { line.                                                         }


     procedure SOUND_ALERT ;

                { This procedure causes an audible signal to be sounded at the }
                { user terminal, often to indicate that the program is waiting }
                { for input from the user, sometimes (in real-time simulators) }
                { to indicate that the program is ignoring an attempted input  }
                { because it is not valid or not capable of timely implementa- }
                { tion in the current circumstance.                             }


     procedure SOUND_ALARM ;

                { This procedure causes a distinctive audible signal to be      }
                { sounded at the user terminal, usually to indicate that some  }
                { error condition has occurred that requires corrective action }
                { on the part of the user.                                      }
```

```
$ page $

      procedure START_NEW_PAGE ;

                { This procedure causes a page-eject character to be written   }
                { to the standard printed-output file LP, which must have been }
                { opened somewhere in the program with a "rewrite" statement.  }


      procedure START_RANDOM_NUMBER_SEQUENCE ( SEED : RANDOMINT ) ;

                { This procedure uses the argument SEED to initiate a repeat-  }
                { able sequence of pseudorandom numbers.                        }


      function  RANDOM_INTEGER : RANDOMINT ;

                { This function returns a pseudorandom integer from a uniform  }
                { distribution in the range of 1 to MAXINT-1 (i.e., in the      }
                { range of 1 to 2147483646).                                    }


      function  CLOCKTICK : integer ;

                { This function returns an integer corresponding to the number }
                { of "ticks" registered on the system clock since some arbi-   }
                { trary fixed time in the past.                                 }


      function  CPUTICK : integer ;

                { This function returns an integer number that grows at the    }
                { rate of the expenditure of CPU time (measured in system      }
                { clock "ticks") to support the calling process.                }


      function  DATESTRING : DATESTR ;

                { This function returns a PASCAL string describing the current }
                { date and time in the form, e.g., "Tue Feb 18 17:32:25 1986". }


      function  NAMESTRING( PAC : NAMEPAC ) : NAMESTR ;

                { This function returns a PASCAL string formed by removing     }
                { any blank spaces found in the input PAC (packed array of     }
                { characters).                                                  }


      function  UPPER_CASE( C : char ) : char ;

                { If C is a lower-case letter of the alphabet this function    }
                { returns its upper-case counterpart;  otherwise it returns    }
                { C unchanged.                                                  }
```

```
$ page $

     function  CAPWORD( WORD : WORDSTR ) : WORDSTR ;

               { This function returns a string formed by replacing all        }
               { lower-case letters in the input WORD (if any) with their      }
               { upper-case counterparts.                                      }


     function  CHAR_INPUT( WMODE : CHWAITMODE ;
                           EMODE : CHECHOMODE ) : CHINPUTREC ;

               { If the user terminal input buffer is empty when this func-    }
               { tion is called, it will will either wait for a character to  }
               { be input or return immediately to the calling routine, de-    }
               { pending on whether the value of WMODE is CHWAIT or NOCHWAIT.  }
               { Whenever it finds the buffer unempty, it removes one char-    }
               { acter and if the value of EMODE is CHECHO it echoes that      }
               { character back to the user terminal.  If WMODE = NOCHWAIT     }
               { and the input buffer is empty then the value of CHAR_INPUT.Q  }
               { is NOTHING; if an end-of-line character is read from the      }
               { buffer (the result of the user having pressed the <RETURN> *  }
               { key) then the value of CHAR_INPUT.Q is ENDOFLINE; otherwise   }
               { the value of CHAR_INPUT.Q is SOMETHING and the value of       }
               { CHAR_INPUT.C is the character that was read from the buffer.  }
               { CHAR_INPUT.C = ' ' (the space character) when the value of    }
               { CHAR_INPUT.Q is NOTHING or ENDOFLINE.                         }


     function  USER_DECIDES_TO( DO_THIS : PROMPTSTR ) : boolean ;

               { The DO_THIS input string should describe a tentative action,  }
               { and after appending the characters ' ? ' it should also form  }
               { a question that can be answered "yes" or "no" by the user     }
               { (with a single keystroke, as explained below).  The boolean   }
               { value of this function is TRUE if the user's answer is "yes"  }
               { and FALSE if the answer is "no".                              }

               { The USER_DECIDES_TO function code causes the DO_THIS string,  }
               { followed by the characters ' ? ', to be displayed on the      }
               { user terminal screen in normal video mode, and then it waits  }
               { for the user to press a single key indicating his decision.   }
               { Pressing 'Y', 'y', or the <RETURN> key indicates "yes".  If   }
               { the user presses 'H','h', or '?' (thinking perhaps to obtain  }
               { a further description of available options), the SOUND_ALARM  }
               { procedure is invoked to give an audible error signal.  Any    }
               { other keystroke is interpreted as "no".  As soon as an an-    }
               { swer is received, an appropriate character string ('YES' or   }
               { 'NO') is displayed in half-bright inverse video mode immed-   }
               { iately behind the prompting question, and the corresponding   }
               { boolean value (TRUE or FALSE) is returned to the calling      }
               { routine.                                                      }
```

```
$ page $

      function  WORD_INPUT( PROMPT   : PROMPTSTR ;
                            DEFAULT  : WORDSTR    ) : WORDSTR   ;

                { The PROMPT string, followed by the characters ' : ', is dis- }
                { played on the user terminal screen in normal video mode and  }
                { followed on the same line by a display of the DEFAULT value  }
                { in blinking bright inverse video mode, signifying that the   }
                { user may approve the blinking value as shown or else supply  }
                { another value (which will replace the one currently blinking }
                { and become the new value to be approved or replaced).  When  }
                { satisfied with the blinking value, the user terminates the   }
                { input process with a single <RETURN> keystroke (i.e., one    }
                { not that is not preceded by any other keystroke).  This      }
                { causes the display mode of the blinking value to be down-    }
                { graded to steady half-bright inverse video, indicating its   }
                { acceptance by the user.  The user-approved value (a charac-  }
                { ter string) is returned (on the stack) to the calling rou-   }
                { tine as the value of the functon WORD_INPUT.                 }

                { If the PROMPT string does not contain a pair of braces (like }
                { the ones enclosing each of these comment lines), then the    }
                { user may type in any character string of his choosing,       }
                { which must be terminated with a <RETURN> keystroke.  After   }
                { deleting any leading or trailing blank spaces, and any other }
                { characters in excess of the number (WORDLEN) required to     }
                { fill the INPUT_WORD function value to its maximum capacity,  }
                { the value thus supplied will replace the previous blinking   }
                { value on the user terminal screen.  This cycle is repeated   }
                { until a blinking value is approved in the fashion described  }
                { in the preceding paragraph.                                  }

                { Braces like the ones enclosing this line --- if present in   }
                { PROMPT --- contain a comma-separated list of the only char-  }
                { acter strings that are permissable input values, from which  }
                { the user must make a choice.  If the blinking value is not   }
                { acceptable to the user, pressing the '-' or the '<' key will }
                { cause its predecessor in the list to be displayed in the     }
                { blinking mode for possible approval; pressing any other key  }
                { except <RETURN> will cause its successor in the list to be   }
                { displayed instead.  Running off the end of the list in       }
                { either direction causes a "wraparound" to the opposite end.  }
                { The maximum number of values allowed in the list is ten.     }


      function  RJWORD_INPUT( PROMPT   : PROMPTSTR ;
                              DEFAULT  : WORDSTR   ;
                              FIELD    : integer   ;
                              MAXLEN   : integer    ) : WORDSTR ;

                { This function works exactly like WORD_INPUT except that the  }
                { the user-input "word", when displayed on the user terminal   }
                { screen, is right-justified in a space at least FIELD columns }
                { wide, and the length of the "word" is constrained to be no   }
                { greater than MAXLEN.                                         }
```

```
$ page $


     function  INTEGER_INPUT( PROMPT  : PROMPTSTR ;
                              DEFAULT : integer   ;
                              FIELD   : integer   ) : integer ;

               { This function works very much like RJWORD_INPUT except of    }
               { course the function value is an integer instead of a string, }
               { which makes the MAXLEN argument of RJWORD_INPUT inappropri-  }
               { ate in this case.  If the user types in a number in fixed-   }
               { point decimal format, it is truncated to make it an integer  }
               { (i.e., the decimal point and any characters following it are }
               { ignored).  Any attempt on the part of the user to input a    }
               { non-numeric value causes an audible error signal to be pro-  }
               { duced by the SOUND_ALARM procedure, and the blinking value   }
               { remains unchanged.                                           }


     function  FIXED_INPUT( PROMPT  : PROMPTSTR ;
                            DEFAULT : longreal  ;
                            FIELD   : integer   ;
                            PLACES  : integer   ) : longreal ;

               { This function is similar to INTEGER_INPUT except that the    }
               { function value is a longreal number instead of an integer.   }
               { A fixed-point format is always used to display the function  }
               { value on the user terminal screen, and the PLACES argument   }
               { specifies how many digits are to be shown after the decimal  }
               { point.  If the user types in a number with more digits than  }
               { what is called for by the PLACES argument, it is rounded to  }
               { the nearest decimal digit in the last place shown on the     }
               { screen.  The same applies to the DEFAULT value supplied      }
               { by the calling routine.  Within the limits of precision that }
               { are inherent in the transformation between the binary and    }
               { the decimal representation of a number, the function value   }
               { returned to the calling routine is exactly that which is     }
               { shown to the user, with no significant decimal digits lurk-  }
               { ing unseen beyond the last one displayed (i.e., "what you    }
               { see is what you get").                                       }

               { This function does not require user input to be typed in a   }
               { fixed-point format.  Integer format is acceptable for whole  }
               { numbers, and it is not necessary to type "0" before the dec- }
               { imal point of a fraction nor behind that of a whole number.  }
               { An exponential format may be used if that is the user's      }
               { preference.  For example, the value displayed on the user    }
               { terminal screen --- and returned to the calling routine ---  }
               { as "150.00" may have been typed in by the user as "149.996", }
               { "150", "150.", "1.5+2", "1.5e2", or "1.499963L2".  If the    }
               { user attempts to input a non-numeric value, FIXED_INPUT will }
               { call the SOUND_ALARM procedure to produce an audible error    }
               { signal, and the blinking value will remain unchanged.        }
```

```
$ page $

implement

     const

          BLINK          = chr( 130 ) ;
          BRIGHTI        = chr( 129 ) ;
          BRIGHTBLINKI   = chr( 131 ) ;
          CLEARTOEOL     = chr( 009 ) ;
          CLEARTOEOM     = chr( 011 ) ;
          FORMFEED       = chr( 012 ) ;
          GOTOCOLZERO    = chr( 013 ) ;
          HALFBRIGHTI    = chr( 137 ) ;
          HOMEUPLEFT     = chr( 001 ) ;
          MOVEUP1ROW     = chr( 031 ) ;
          NORMAL         = chr( 128 ) ;
          UNDERLINE      = chr( 132 ) ;

     type

          STRING3 = string [ 3 ] ;

     var

          RANDOMSEED : RANDOMINT ;
```

```
$ page $


procedure INITIALIZE_IO ;

     begin
     Ioinitialize ;
     RANDOMSEED := IMAX( 1, IMIN( maxint-1, CLOCKTICK ) ) ;
     end ;


procedure CLEAN_UP_IO ;

     begin
     Iouninitialize ;
     end ;


procedure CLEAR_LINE ;

     begin
     write ( GOTOCOLZERO,CLEARTOEOL ) ;
     end ;


procedure CLEAR_SCREEN ;

     begin
     write ( HOMEUPLEFT,CLEARTOEOM ) ;
     end ;


procedure FETCHLN ( var STR : LINESTR ) ;

     begin
     readln ( STR ) ;
     end ;


procedure LOITER ( MILLISECS : integer ) ;

     var

          TICK : integer ;
          TOCK : integer ;

     begin
     TOCK := Sysclock + ( TICKSPERSEC * MILLISECS ) div 1000 ;
     repeat
          TICK := Sysclock   ;
          until TICK > TOCK ;
     end ;
```

```
$ page $

procedure MOVE_UP ;

     begin
     write ( MOVEUP1ROW ) ;
     end ;


procedure SET_BRIGHTBLINK_INVERSE_VIDEO ;

     begin
     write ( BRIGHTBLINKI ) ;
     end ;


procedure SET_HALFBRIGHT_INVERSE_VIDEO ;

     begin
     write ( HALFBRIGHTI ) ;
     end ;


procedure SET_NORMAL_VIDEO ;

     begin
     write ( NORMAL ) ;
     end ;


procedure SHOW ( STR : LINESTR ) ;

     begin
     write ( STR ) ;
     end ;


procedure SHOWLN ( STR : LINESTR ) ;

     begin
     writeln ( STR ) ;
     end ;


procedure SOUND_ALERT ;

     begin
     Beep ;
     end ;


procedure SOUND_ALARM ;

     begin
     Beeper ( 3, 50 ) ;
     end ;
```

```
$ page $


procedure START_NEW_PAGE ;

     begin
     write ( LP, FORMFEED ) ;
     end ;


procedure START_RANDOM_NUMBER_SEQUENCE ( SEED : RANDOMINT ) ;

     begin
     RANDOMSEED := SEED ;
     end ;


function RANDOM_INTEGER : RANDOMINT ;

     var

          I : integer ;

     begin
     I := RANDOMSEED ;
     Random ( I ) ;
     RANDOMSEED := I ;
     RANDOM_INTEGER := I ;
     end ;


function CLOCKTICK : integer ;

     begin
     CLOCKTICK := Sysclock ;
     end ;


function CPUTICK : integer ;

     begin
     CPUTICK := Sysclock ;
     end ;
```

```
$ page $


function DATESTRING : DATESTR ;

     type

          DAYARR = array [ 0..6 ]  of STRING3 ;
          MONARR = array [ 1..12 ] of STRING3 ;

     const

          DAYNAME = DAYARR [ 'Sun','Mon','Tue','Wed','Thu','Fri','Sat' ] ;
          MONNAME = MONARR [ 'Jan','Feb','Mar','Apr','May','Jun',
                             'Jul','Aug','Sep','Oct','Nov','Dec' ] ;

     var

          D      : integer ;
          DATE   : Daterec ;
          K      : integer ;
          JD     : integer ;
          SECOND : integer ;
          TIME   : Timerec ;
          WORK   : DATESTR ;
          YR     : integer ;

     begin
     Sysdate ( DATE ) ;
     Systime ( TIME ) ;
     with DATE,TIME do
          begin
          SECOND := round( Centisecond / 100 ) ;
          YR := 1900 + Year ;
          JD := JULIAN_DAYNUM( YR, Month, Day) ;
          D := ( JD + 1 ) mod 7 ;
          WORK := '' ;
          strwrite ( WORK, 1, K, DAYNAME[D],' ' ) ;
          strwrite ( WORK, K, K, MONNAME[Month],' ' ) ;
          if Day < 10
               then strwrite ( WORK, K, K, '0',Day:1,' ' )
               else strwrite ( WORK, K, K, Day:2,' ' ) ;
          if Hour < 10
               then strwrite ( WORK, K, K, '0',Hour:1,':' )
               else strwrite ( WORK, K, K, Hour:2,':' ) ;
          if Minute < 10
               then strwrite ( WORK, K, K, '0',Minute:1,':' )
               else strwrite ( WORK, K, K, Minute:2,':' ) ;
          if SECOND < 10
               then strwrite ( WORK, K, K, '0',SECOND:1,' ' )
               else strwrite ( WORK, K, K, SECOND:2,' ' ) ;
          strwrite ( WORK, K, K, YR:4 ) ;
          end ;
     DATESTRING := WORK ;
     end ;
```

```
$ page $


function NAMESTRING ( PAC : NAMEPAC ) : NAMESTR ;

     var

          i    : integer ;
          n    : integer ;
          WORK : NAMESTR ;

     begin
     setstrlen( WORK, NAMELEN ) ;
     n := 0 ;
     for i := 1 to NAMELEN do
          if PAC[i] <> ' ' then
               begin
               n := n + 1 ;
               WORK[n] := PAC[i] ;
               end ;
     setstrlen ( WORK, n ) ;
     NAMESTRING := WORK ;
     end ;


function UPPER_CASE( C : char ) : char ;

     var

          K : integer ;

     begin
     K := ord( C ) ;
     if K >= ord( 'a' ) then
          if K <= ord( 'z' ) then
                C := chr( K + ord( 'A' ) - ord( 'a' ) ) ;
     UPPER_CASE := C ;
     end ;


function CAPWORD( WORD : WORDSTR ) : WORDSTR ;

     var

          i : integer ;
          n : integer ;

     begin
     n := strlen( WORD ) ;
     if n > 0 then
          for i := 1 to n do
               WORD[i] := UPPER_CASE( WORD[i] ) ;
     CAPWORD := WORD ;
     end ;
```

```
$ page $


function CHAR_INPUT( WMODE : CHWAITMODE ;
                     EMODE : CHECHOMODE ) : CHINPUTREC ;

     var

          S    : string [ 1 ] ;
          WORK : CHINPUTREC   ;

     begin
     case WMODE of

          NOCHWAIT :
          begin
          if Keybuffer^.Size = 0

               then WORK.Q := NOTHING

               else
               begin
               Keybufops ( Kgetchar, WORK.C ) ;
               if WORK.C = chr( 013 )
                    then WORK.Q := ENDOFLINE
                    else WORK.Q := SOMETHING ;
               end ;

          end ;

          CHWAIT :
          begin
          while Keybuffer^.Size = 0 do
               WORK.C := ' ' ;
          Keybufops ( Kgetchar, WORK.C ) ;
          if WORK.C = chr( 013 )
               then WORK.Q := ENDOFLINE
               else WORK.Q := SOMETHING ;
          end ;

          end ; { case WMODE }
     if WORK.Q <> SOMETHING then
          WORK.C := ' ' ;
     if EMODE = CHECHO then
          begin
          setstrlen ( S, 1 ) ;
          S[1] := WORK.C ;
          SHOW ( S ) ;
          end ;
     CHAR_INPUT := WORK ;
     end ;
```

```
$ page $


function USER_DECIDES_TO( DO_THIS : PROMPTSTR ) : boolean ;

     var

          ANSWER      : string[ 3 ] ;
          OKAY        : boolean     ;
          WORK        : CHINPUTREC  ;


     begin
     SOUND_ALERT ;
     repeat
          SHOW ( DO_THIS+' ? ' ) ;
          WORK := CHAR_INPUT( CHWAIT, NOCHECHO ) ;
          OKAY := true ;
          if WORK.Q = ENDOFLINE

               then ANSWER := 'YES'

               else
               begin
               case WORK.C of

                    'Y','y':
                    ANSWER := 'YES' ;

                    'H','h','?':
                    begin
                    OKAY := false ;
                    SOUND_ALARM ;
                    end ;

                    otherwise
                    ANSWER :=  'NO' ;

                    end ; { case WORK.C }
               end ;

          CLEAR_LINE ;
          until OKAY ;
     SHOW ( DO_THIS+' ? ' ) ;
     SET_HALFBRIGHT_INVERSE_VIDEO ;
     SHOW ( ' '+ANSWER+' ' ) ;
     SET_NORMAL_VIDEO ;
     SHOWLN ( '' ) ;
     if ANSWER = 'NO'
          then USER_DECIDES_TO { DO_THIS } := false
          else USER_DECIDES_TO { DO_THIS } := true ;
     end ;
```

```
$ page $


function  WORD_INPUT( PROMPT   : PROMPTSTR  ;
                      DEFAULT : WORDSTR    ) : WORDSTR    ;

     begin
     WORD_INPUT := RJWORD_INPUT( PROMPT, DEFAULT, 0, WORDLEN ) ;
     end ;



function  RJWORD_INPUT( PROMPT   : PROMPTSTR   ;
                        DEFAULT : WORDSTR    ;
                        FIELD   : integer    ;
                        MAXLEN  : integer    ) : WORDSTR    ;

     var

          base              : integer ;
          current           : integer ;
          DELIM             : integer ;
          FLD               : integer ;
          K                 : integer ;
          LBRACE            : integer ;
          LEN               : integer ;
          OPTION            : array [ 0..9 ] of WORDSTR ;
          RBRACE            : integer ;
          SELECT_MODE       : boolean ;
          VALUE             : WORDSTR ;
          VALUE_APPROVED    : boolean ;
          WERK              : CHINPUTREC ;
          WIRK              : LINESTR ;
          WORK              : LINESTR ;
```

```
$ page $

     begin { function RJWORD_INPUT }
     MAXLEN := IMIN( WORDLEN, MAXLEN ) ;
     WORK := strltrim( strrtrim( DEFAULT ) ) ;
     LEN := IMIN( MAXLEN, strlen( WORK ) ) ;
     DEFAULT := str( WORK, 1, LEN ) ;
     SELECT_MODE := false ;
     LBRACE := strpos( PROMPT, '{' ) ;
     if LBRACE > 0 then
          begin
          RBRACE := strpos( PROMPT, '}' ) ;
          if RBRACE > LBRACE then
               begin
               SELECT_MODE := true ;
               base := 0 ;
               LEN := RBRACE - LBRACE ;
               WORK := str( PROMPT, LBRACE+1, LEN ) ;
               repeat
                    DELIM := strpos( WORK, ',' ) ;
                    if DELIM = 0 then
                         DELIM := strpos( WORK, '}' ) ;
                    LEN := DELIM - 1 ;
                    WIRK := strltrim( strrtrim( str( WORK, 1, LEN ) ) ) ;
                    LEN := IMIN( MAXLEN, strlen( WIRK ) ) ;
                    OPTION[base] := str( WIRK, 1, LEN ) ;
                    strdelete( WORK, 1, DELIM ) ;
                    base := base + 1 ;
                    until strlen( WORK ) = 0 ;
               current := base ;
               repeat
                    current := current - 1 ;
                    until ( OPTION[current] = DEFAULT ) or ( current = 0 ) ;
               DEFAULT := OPTION[current] ;
               end ;
          end ;
     VALUE := DEFAULT ;
     SOUND_ALERT ;
     repeat
          SHOW ( PROMPT+' : ' ) ;
          SET_BRIGHTBLINK_INVERSE_VIDEO ;
          FLD := IMAX( FIELD, strlen( VALUE ) ) ;
          WORK := '' ;
          strwrite ( WORK,1,K,' ',VALUE:FLD,' ' ) ;
          SHOW ( WORK ) ;
          SET_NORMAL_VIDEO ;
```

```
$ page $

          if SELECT_MODE

               then
               begin
               WERK := CHAR_INPUT( CHWAIT, NOCHECHO ) ;
               if WERK.Q = ENDOFLINE

                    then VALUE_APPROVED := true

                    else
                    begin
                    VALUE_APPROVED := false ;
                    if WERK.C in [ '<','-' ]
                         then current := ( current - 1 ) mod base
                         else current := ( current + 1 ) mod base ;
                    VALUE := OPTION[current] ;
                    end ;

               CLEAR_LINE ;
               end

               else
               begin
               SHOWLN ( '' ) ;
               FETCHLN ( WORK ) ;
               MOVE_UP ;
               CLEAR_LINE ;
               MOVE_UP ;
               CLEAR_LINE ;
               if strlen( WORK ) = 0

                    then VALUE_APPROVED := true

                    else
                    begin
                    VALUE_APPROVED := false ;
                    WORK := strltrim( strrtrim( WORK ) ) ;
                    LEN := strlen( WORK ) ;
                    if LEN > MAXLEN

                         then
                         begin
                         LEN := MAXLEN ;
                         SOUND_ALARM ;
                         end

                         else SOUND_ALERT ;

                    VALUE := str( WORK, 1, LEN ) ;
                    end ;

               end ;

          until VALUE_APPROVED ;

     SHOW ( PROMPT+' : ' ) ;
     SET_HALFBRIGHT_INVERSE_VIDEO ;
     WORK := '' ;
```

```
    strwrite ( WORK,1,K,' ',VALUE:FLD,' ' ) ;
    SHOW ( WORK ) ;
    SET_NORMAL_VIDEO ;
    SHOWLN ( '' ) ;
    RJWORD_INPUT := VALUE ;
    end ; { function RJWORD_INPUT }
```

```
$ page $


function INTEGER_INPUT( PROMPT  : PROMPTSTR ;
                        DEFAULT : integer   ;
                        FIELD   : integer   ) : integer ;

     var

          K                 : integer ;
          TRIAL_VALUE       : integer ;
          VALUE             : integer ;
          VALUE_APPROVED    : boolean ;
          WORK              : LINESTR ;

     begin
     VALUE := DEFAULT ;
     SOUND_ALERT ;
     repeat
          SHOW ( PROMPT+' : ' ) ;
          SET_BRIGHTBLINK_INVERSE_VIDEO ;
          WORK := '' ;
          strwrite ( WORK,1,K,' ',VALUE:FIELD,' ' ) ;
          SHOW ( WORK ) ;
          SET_NORMAL_VIDEO ;
          SHOWLN ( '' ) ;
          FETCHLN ( WORK ) ;
          MOVE_UP ;
          CLEAR_LINE ;
          MOVE_UP ;
          CLEAR_LINE ;
          if strlen( WORK ) = 0

               then
               VALUE_APPROVED := true

               else
               begin
               VALUE_APPROVED := false ;
               try                                    { set trap for possible error }
                    strread ( WORK, 1, K, TRIAL_VALUE ) ;   { error here maybe }
                    VALUE := TRIAL_VALUE ;
                    SOUND_ALERT ;        { and jump around "recover" statement }
                    recover SOUND_ALARM ;      { come here after error, if any }
               end ;

          until VALUE_APPROVED ;
     SHOW ( PROMPT+' : ' ) ;
     SET_HALFBRIGHT_INVERSE_VIDEO ;
     WORK := '' ;
     strwrite ( WORK,1,K,' ',VALUE:FIELD,' ' ) ;
     SHOW ( WORK ) ;
     SET_NORMAL_VIDEO ;
     SHOWLN ( '' ) ;
     INTEGER_INPUT := VALUE ;
     end ;
```

```
$ page $


function FIXED_INPUT( PROMPT   : PROMPTSTR ;
                      DEFAULT  : longreal ;
                      FIELD    : integer  ;
                      PLACES   : integer  ) : longreal ;

     var

          K                : integer  ;
          TRIAL_VALUE      : longreal ;
          VALUE            : longreal ;
          VALUE_APPROVED   : boolean  ;
          WORK             : LINESTR  ;

     begin
     VALUE := DEFAULT ;
     SOUND_ALERT ;
     repeat
          WORK := '' ;
          strwrite ( WORK, 1, K, VALUE:FIELD:PLACES ) ;
          strread  ( WORK, 1, K, VALUE ) ;
          SHOW ( PROMPT+' : ' ) ;
          SET_BRIGHTBLINK_INVERSE_VIDEO ;
          SHOW ( ' '+WORK+' ' ) ;
          SET_NORMAL_VIDEO ;
          SHOWLN ( '' ) ;
          FETCHLN ( WORK ) ;
          MOVE_UP ;
          CLEAR_LINE ;
          MOVE_UP ;
          CLEAR_LINE ;
          if strlen( WORK ) = 0

               then
               VALUE_APPROVED := true

               else
               begin
               VALUE_APPROVED := false ;
               try                                  { set trap for possible error }
                    strread ( WORK, 1, K, TRIAL_VALUE ) ;  { error here maybe }
                    VALUE := TRIAL_VALUE ;
                    SOUND_ALERT ;        { and jump around "recover" statement }
                    recover SOUND_ALARM ;      { come here after error, if any }
               end ;

          until VALUE_APPROVED ;
     SHOW ( PROMPT+' : ' ) ;
     SET_HALFBRIGHT_INVERSE_VIDEO ;
     WORK := '' ;
     strwrite ( WORK,1,K,' ',VALUE:FIELD:PLACES,' ' ) ;
     SHOW ( WORK ) ;
     SET_NORMAL_VIDEO ;
     SHOWLN ( '' ) ;
     FIXED_INPUT := VALUE ;
     end ;
```

```
end ;  { module UTILSPIF & File 'UTILSPIF.I' }
```

```
$ page $  { begin File 'utilspif.I' }

{ Utility Software Unit for HP-9000 Series 500 with HP-UX 5.0 Op Sys }

module UTILSPIF ;  { Subject  :  System/PASCAL Interface }
                   { Domain   :  Universal               }

                   { NASA/JSC/MPAD/TRW      Sam Wilson }
                   { Updated  Thu Apr 10 23:16:19 1986 }

                   { This one module contains all of the system-dependent    }
                   { data and PASCAL code needed for most applications.      }

import

     UTILMATH ;

export

     const

          TICKSPERSEC =   60     ;              { Series 500 clock resolution }

          DATELEN   = 24 ; { number of characters in date-and-time string    }
          LINELEN   = 80 ; { max characters in input/output line string      }
          NAMELEN   =  8 ; { max characters in "name" string                 }
          PROMPTLEN = 60 ; { max characters in prompt string for user input  }
          WORDLEN   = 20 ; { max characters in "word" string                 }
```

```
$ page $

     type

          RANDOMINT = 1..maxint-1 ;     { pseudorandom integer : 1..2147483646 }

          DATESTR   = string [ DATELEN   ] ;
          LINESTR   = string [ LINELEN   ] ;
          NAMEPAC   = packed array [ 1..NAMELEN ]  of char ;
          NAMESTR   = string [ NAMELEN   ] ;
          PROMPTSTR = string [ PROMPTLEN ] ;
          WORDSTR   = string [ WORDLEN   ] ;

          CHWAITMODE = (     { describes wait mode for CHAR_INPUT function      }
               CHWAIT,       { wait if input buffer is empty                    }
               NOCHWAIT ) ;  { don't wait if input buffer is empty              }

          CHECHOMODE = (     { describes echo mode for CHAR_INPUT function      }
               CHECHO,       { echo characters back to user terminal screen     }
               NOCHECHO ) ;  { don't echo characters back to user terminal      }

          GOTWHAT = (        { describes result of executing CHAR_INPUT functn }
               NOTHING,      { the input buffer was empty (& didn't wait)       }
               ENDOFLINE,    { got an end-of-line char (<RETURN> keystroke)     }
               SOMETHING );  { got a character that was not end-of-line         }

          CHINPUTREC =
               record  { returned by the CHAR_INPUT function }
               Q : GOTWHAT ;        { qualifier for the returned character      }
               C : char    ;        { C = ' ' if Q = NOTHING or ENDOFLINE;      }
               end ; { record }     { else C = char produced by user keystroke }


     var

          LP      : text     ; { standard file for printed output; must be      }
                               { opened by "rewrite" statement somewhere in     }
                               { program before use                             }
```

```
$ page $

     procedure INITIALIZE_IO ;

               { This procedure performs the I/O system initialization re-     }
               { quired to support most applications.  Normally it should be   }
               { the first procedure called by any program.                     }


     procedure CLEAN_UP_IO ;

               { This procedure closes any open files and generally cleans     }
               { up the I/O system.  Normally it should be the last procedure }
               { called by any program.                                         }


     procedure CLEAR_LINE ;

               { This procedure clears the line on the user terminal where     }
               { the cursor is currently located.                               }


     procedure CLEAR_SCREEN ;

               { This procedure clears the user terminal screen.                }


     procedure FETCHLN ( var STR : LINESTR ) ;

               { This procedure reads (and thus removes) characters from the  }
               { user terminal input buffer --- storing them in the string     }
               { variable STR --- until an end-of-line character (generated    }
               { by a <RETURN>* keystroke) has been read.  The end-of-line     }
               { character is always discarded, along with any other charac-  }
               { ters in excess of the number (LINELEN) needed to fill the     }
               { variable STR to its maximum capacity.                          }

               { * NOTE: Some keyboards, instead of having a <RETURN> key,      }
               {         have an <ENTER> key that serves the same purpose.      }
               {         Wherever "<RETURN>" appears in this and following      }
               {         descriptions, it should be interpreted as "<RETURN>   }
               {         or <ENTER>, whichever is present on the keyboard".    }


     procedure LOITER ( MILLISECS : integer ) ;

               { This procedure loops through meaningless code (effectively   }
               { suspending program execution) for the number of milliseconds }
               { specified in its argument.                                     }
```

```
$ page $

     procedure MOVE_UP ;

               { This procedure moves the cursor up one line on the user      }
               { terminal.                                                    }


     procedure SET_BRIGHTBLINK_INVERSE_VIDEO ;

               { This procedure causes characters to be written on the user   }
               { terminal screen in the blinking bright inverse video mode,   }
               { which remains in effect until another mode is specified.     }


     procedure SET_HALFBRIGHT_INVERSE_VIDEO ;

               { This procedure causes characters to be written on the user   }
               { terminal screen in the halfbright inverse video mode,        }
               { which remains in effect until another mode is specified.     }


     procedure SET_NORMAL_VIDEO ;

               { This procedure causes characters to be written on the user   }
               { terminal screen in the normal video mode, which remains in   }
               { effect until another mode is specified.                      }


     procedure SHOW ( STR : LINESTR ) ;

               { This procedure writes a PASCAL string to the user terminal.  }


     procedure SHOWLN ( STR : LINESTR ) ;

               { This procedure writes a PASCAL string to the user terminal,  }
               { and then positions the cursor at the beginning of the next   }
               { line.                                                        }


     procedure SOUND_ALERT ;

               { This procedure causes an audible signal to be sounded at the }
               { user terminal, often to indicate that the program is waiting }
               { for input from the user, sometimes (in real-time simulators) }
               { to indicate that the program is ignoring an attempted input  }
               { because it is not valid or not capable of timely implementa- }
               { tion in the current circumstance.                            }


     procedure SOUND_ALARM ;

               { This procedure causes a distinctive audible signal to be     }
               { sounded at the user terminal, usually to indicate that some  }
               { error condition has occurred that requires corrective action }
               { on the part of the user.                                     }
```

```
$ page $


     procedure START_NEW_PAGE ;

               { This procedure causes a page-eject character to be written   }
               { to the standard printed-output file LP, which must have been }
               { opened somewhere in the program with a "rewrite" statement.  }


     procedure START_RANDOM_NUMBER_SEQUENCE ( SEED : RANDOMINT ) ;

               { This procedure uses the argument SEED to initiate a repeat-  }
               { able sequence of pseudorandom numbers.                       }


     function  RANDOM_INTEGER : RANDOMINT ;

               { This function returns a pseudorandom integer from a uniform  }
               { distribution in the range of 1 to MAXINT-1 (i.e., in the     }
               { range of 1 to 2147483646).                                   }


     function  CLOCKTICK : integer ;

               { This function returns an integer corresponding to the number }
               { of "ticks" registered on the system clock since some arbi-   }
               { trary fixed time in the past.                                }


     function  CPUTICK : integer ;

               { This function returns an integer number that grows at the    }
               { rate of the expenditure of CPU time (measured in system      }
               { clock "ticks") to support the calling process.               }


     function  DATESTRING : DATESTR ;

               { This function returns a PASCAL string describing the current }
               { date and time in the form, e.g., "Tue Feb 18 17:32:25 1986". }


     function  NAMESTRING( PAC : NAMEPAC ) : NAMESTR ;

               { This function returns a PASCAL string formed by removing     }
               { any blank spaces found in the input PAC (packed array of     }
               { characters).                                                 }


     function  UPPER_CASE( C : char ) : char ;

               { If C is a lower-case letter of the alphabet this function    }
               { returns its upper-case counterpart;  otherwise it returns    }
               { C unchanged.                                                 }
```

```
$ page $


     function  CAPWORD( WORD : WORDSTR ) : WORDSTR ;

               { This function returns a string formed by replacing all        }
               { lower-case letters in the input WORD (if any) with their      }
               { upper-case counterparts.                                      }


     function  CHAR_INPUT( WMODE : CHWAITMODE ;
                           EMODE : CHECHOMODE ) : CHINPUTREC ;

               { If the user terminal input buffer is empty when this func-    }
               { tion is called, it will will either wait for a character to   }
               { be input or return immediately to the calling routine, de-    }
               { pending on whether the value of WMODE is CHWAIT or NOCHWAIT.  }
               { Whenever it finds the buffer unempty, it removes one char-    }
               { acter and if the value of EMODE is CHECHO it echoes that      }
               { character back to the user terminal.  If WMODE = NOCHWAIT     }
               { and the input buffer is empty then the value of CHAR_INPUT.Q  }
               { is NOTHING; if an end-of-line character is read from the      }
               { buffer (the result of the user having pressed the <RETURN>    }
               { key) then the value of CHAR_INPUT.Q is ENDOFLINE; otherwise   }
               { the value of CHAR_INPUT.Q is SOMETHING and the value of       }
               { CHAR_INPUT.C is the character that was read from the buffer.  }
               { CHAR_INPUT.C = ' ' (the space character) when the value of    }
               { CHAR_INPUT.Q is NOTHING or ENDOFLINE.                         }


     function  USER_DECIDES_TO( DO_THIS : PROMPTSTR ) : boolean ;

               { The DO_THIS input string should describe a tentative action,  }
               { and after appending the characters ' ? ' it should also form  }
               { a question that can be answered "yes" or "no" by the user     }
               { (with a single keystroke, as explained below).  The boolean   }
               { value of this function is TRUE if the user's answer is "yes"  }
               { and FALSE if the answer is "no".                              }

               { The USER_DECIDES_TO function code causes the DO_THIS string,  }
               { followed by the characters ' ? ', to be displayed on the      }
               { user terminal screen in normal video mode, and then it waits  }
               { for the user to press a single key indicating his decision.   }
               { Pressing 'Y', 'y', or the <RETURN> key indicates "yes".  If   }
               { the user presses 'H','h', or '?' (thinking perhaps to obtain  }
               { a further description of available options), the SOUND_ALARM  }
               { procedure is invoked to give an audible error signal.  Any    }
               { other keystroke is interpreted as "no".  As soon as an an-    }
               { swer is received, an appropriate character string ('YES' or   }
               { 'NO') is displayed in half-bright inverse video mode immed-   }
               { iately behind the prompting question, and the corresponding   }
               { boolean value (TRUE or FALSE) is returned to the calling      }
               { routine.                                                      }
```

```
$ page $

    function  WORD_INPUT( PROMPT   : PROMPTSTR ;
                          DEFAULT  : WORDSTR    ) : WORDSTR   ;

              { The PROMPT string, followed by the characters ' : ', is dis- }
              { played on the user terminal screen in normal video mode and  }
              { followed on the same line by a display of the DEFAULT value  }
              { in blinking bright inverse video mode, signifying that the   }
              { user may approve the blinking value as shown or else supply  }
              { another value (which will replace the one currently blinking }
              { and become the new value to be approved or replaced).  When  }
              { satisfied with the blinking value, the user terminates the   }
              { input process with a single <RETURN> keystroke (i.e., one    }
              { not that is not preceded by any other keystroke).  This      }
              { causes the display mode of the blinking value to be down-    }
              { graded to steady half-bright inverse video, indicating its   }
              { acceptance by the user.  The user-approved value (a charac-  }
              { ter string) is returned (on the stack) to the calling rou-   }
              { tine as the value of the functon WORD_INPUT.                 }

              { If the PROMPT string does not contain a pair of braces (like }
              { the ones enclosing each of these comment lines), then the    }
              { user may type in any character string of his choosing,       }
              { which must be terminated with a <RETURN> keystroke.  After   }
              { deleting any leading or trailing blank spaces, and any other }
              { characters in excess of the number (WORDLEN) required to     }
              { fill the INPUT_WORD function value to its maximum capacity,  }
              { the value thus supplied will replace the previous blinking   }
              { value on the user terminal screen.  This cycle is repeated   }
              { until a blinking value is approved in the fashion described  }
              { in the preceding paragraph.                                  }

              { Braces like the ones enclosing this line --- if present in   }
              { PROMPT --- contain a comma-separated list of the only char-  }
              { acter strings that are permissable input values, from which  }
              { the user must make a choice.  If the blinking value is not   }
              { acceptable to the user, pressing the '-' or the '<' key will }
              { cause its predecessor in the list to be displayed in the     }
              { blinking mode for possible approval; pressing any other key  }
              { except <RETURN> will cause its successor in the list to be   }
              { displayed instead.  Running off the end of the list in       }
              { either direction causes a "wraparound" to the opposite end.  }
              { The maximum number of values allowed in the list is ten.     }


    function  RJWORD_INPUT( PROMPT   : PROMPTSTR ;
                            DEFAULT  : WORDSTR   ;
                            FIELD    : integer   ;
                            MAXLEN   : integer    ) : WORDSTR ;

              { This function works exactly like WORD_INPUT except that the  }
              { the user-input "word", when displayed on the user terminal   }
              { screen, is right-justified in a space at least FIELD columns }
              { wide, and the length of the "word" is constrained to be no   }
              { greater than MAXLEN.                                         }
```

```
$ page $

      function  INTEGER_INPUT( PROMPT  : PROMPTSTR ;
                               DEFAULT : integer   ;
                               FIELD   : integer   ) : integer ;

          { This function works very much like RJWORD_INPUT except of     }
          { course the function value is an integer instead of a string,  }
          { which makes the MAXLEN argument of RJWORD_INPUT inappropri-   }
          { ate in this case.  If the user types in a number in fixed-    }
          { point decimal format, it is truncated to make it an integer   }
          { (i.e., the decimal point and any characters following it are  }
          { ignored).  Any attempt on the part of the user to input a     }
          { non-numeric value causes an audible error signal to be pro-   }
          { duced by the SOUND_ALARM procedure, and the blinking value    }
          { remains unchanged.                                            }


      function  FIXED_INPUT( PROMPT  : PROMPTSTR ;
                             DEFAULT : longreal  ;
                             FIELD   : integer   ;
                             PLACES  : integer   ) : longreal ;

          { This function is similar to INTEGER_INPUT except that the     }
          { function value is a longreal number instead of an integer.    }
          { A fixed-point format is always used to display the function   }
          { value on the user terminal screen, and the PLACES argument    }
          { specifies how many digits are to be shown after the decimal   }
          { point.  If the user types in a number with more digits than   }
          { what is called for by the PLACES argument, it is rounded to   }
          { the nearest decimal digit in the last place shown on the      }
          { screen.  The same applies to the DEFAULT value supplied       }
          { by the calling routine.  Within the limits of precision that  }
          { are inherent in the transformation between the binary and     }
          { the decimal representation of a number, the function value    }
          { returned to the calling routine is exactly that which is      }
          { shown to the user, with no significant decimal digits lurk-   }
          { ing unseen beyond the last one displayed (i.e., "what you     }
          { see is what you get").                                        }

          { This function does not require user input to be typed in a    }
          { fixed-point format.  Integer format is acceptable for whole   }
          { numbers, and it is not necessary to type "0" before the dec-  }
          { imal point of a fraction nor behind that of a whole number.   }
          { An exponential format may be used if that is the user's       }
          { preference.  For example, the value displayed on the user     }
          { terminal screen --- and returned to the calling routine ---   }
          { as "150.00" may have been typed in by the user as "149.996",  }
          { "150", "150.", "1.5+2", "1.5e2", or "1.499963L2".  If the     }
          { user attempts to input a non-numeric value, FIXED_INPUT will  }
          { call the SOUND_ALARM procedure to produce an audible error    }
          { signal, and the blinking value will remain unchanged.         }
```

```
$ page $

implement

     const

          DATEPACLEN = 26 ;
          FORMFEED   = chr( 012 ) ;
          NEWLINE    = chr( 010 ) ;
          NULLCHAR   = chr( 000 ) ;

     type

          DATEPAC = packed array [ 1..DATEPACLEN ] of char ;
          DATEPTR = ^DATEPAC ;
          PAC256  = packed array [ 1..256 ] of char ;
          TMSREC  =
               record
               UTIME  : integer ;
               STIME  : integer ;
               CUTIME : integer ;
               CSTIME : integer ;
               end ; { record }

     var

          RANDOMSEED : RANDOMINT ;


{    HP-UX 5.0 system routines called from this module:                         }

     function  ctime        ( var SECONDS : integer ) : DATEPTR  ; external ;
     function  times        ( var TMS     : TMSREC  ) : integer  ; external ;
     procedure exit         (     STATUS  : integer )            ; external ;
     procedure time         ( var SECONDS : integer )            ; external ;


{    C routines (defined in 'utilscif.c') called from this module:              }

     procedure clearline                                         ; external ;
     procedure clearscreen                                       ; external ;
     procedure fetchpac     ( var PAC : PAC256 )                 ; external ;
     procedure ioinitialize                                      ; external ;
     procedure iouninitialize                                    ; external ;
     procedure moveup1row                                        ; external ;
     procedure setbrightblinki                                   ; external ;
     procedure setcbreak                                         ; external ;
     procedure setcecho                                          ; external ;
     procedure setcwait                                          ; external ;
     procedure sethalfbrighti                                    ; external ;
     procedure setnocbreak                                       ; external ;
     procedure setnocecho                                        ; external ;
     procedure setnocwait                                        ; external ;
     procedure setnormalvideo                                    ; external ;
     procedure showpac      ( var PAC : PAC256 )                 ; external ;
     procedure soundalert                                        ; external ;
     function  kbdcharcode                    : integer          ; external ;
```

```
$ page $


procedure INITIALIZE_IO ;

     begin
     ioinitialize ;
     RANDOMSEED := IMAX( 1, IMIN( maxint-1, CLOCKTICK ) ) ;
     end ;


procedure CLEAN_UP_IO ;

     begin
     iouninitialize ;
     end ;


procedure CLEAR_LINE ;

     begin
     clearline ;
     end ;


procedure CLEAR_SCREEN ;

     begin
     clearscreen ;
     end ;


procedure FETCHLN ( var STR : LINESTR ) ;

     var

          i    : integer ;
          WORK : PAC256  ;

     begin
     fetchpac ( WORK ) ;
     setstrlen( STR, 0 ) ;
     i := 0 ;
     repeat
          i := i + 1 ;
          if WORK[i] <> NULLCHAR then
               begin
               setstrlen ( STR, i ) ;
               STR[i] := WORK[i] ;
               end ;
          until ( WORK[i] = NULLCHAR ) or ( i = LINELEN ) ;
     end ;
```

```
$ page $


procedure LOITER ( MILLISECS : integer ) ;

     var

          TICK : integer ;
          TMS  : TMSREC  ;
          TOCK : integer ;

     begin
     TOCK := times( TMS ) + ( TICKSPERSEC * MILLISECS ) div 1000 ;
     repeat
          TICK := times( TMS ) ;
          until TICK > TOCK ;
     end ;


procedure MOVE_UP ;

     begin
     moveup1row ;
     end ;


procedure SET_BRIGHTBLINK_INVERSE_VIDEO ;

     begin
     setbrightblinki ;
     end ;


procedure SET_HALFBRIGHT_INVERSE_VIDEO ;

     begin
     sethalfbrighti ;
     end ;


procedure SET_NORMAL_VIDEO ;

     begin
     setnormalvideo ;
     end ;
```

```
$ page $


procedure SHOW ( STR : LINESTR ) ;

     var

          i    : integer ;
          n    : integer ;
          WORK : PAC256  ;

     begin
     n := strlen( STR ) ;
     if n > 0 then
          for i := 1 to n do
               WORK[i] := STR[i] ;
     WORK[n+1] := NULLCHAR ;
     showpac ( WORK ) ;
     end ;


procedure SHOWLN ( STR : LINESTR ) ;

     var

          i    : integer ;
          n    : integer ;
          WORK : PAC256  ;

     begin
     n := strlen( STR ) ;
     if n > 0 then
          for i := 1 to n do
               WORK[i] := STR[i] ;
     WORK[n+1] := NEWLINE  ;
     WORK[n+2] := NULLCHAR ;
     showpac ( WORK ) ;
     end ;


procedure SOUND_ALERT ;

     begin
     soundalert ;
     end ;


procedure SOUND_ALARM ;

     begin
     SOUND_ALERT ;
     LOITER ( 400 ) ;
     SOUND_ALERT ;
     LOITER ( 200 ) ;
     SOUND_ALERT ;
     end ;
```

```
$ page $


procedure START_NEW_PAGE ;

     begin
     write ( LP, FORMFEED ) ;
     end ;


procedure START_RANDOM_NUMBER_SEQUENCE ( SEED : RANDOMINT ) ;

     begin
     RANDOMSEED := SEED ;
     end ;


function RANDOM_INTEGER : RANDOMINT ;

     begin  { emulation of Series 200 pseudorandom number generator }
     RANDOMSEED := trunc( RMOD( ( RANDOMSEED * 16807L0 ), maxint ) ) ;
     RANDOM_INTEGER := RANDOMSEED ;
     end ;


function CLOCKTICK : integer ;

     var

          TMS : TMSREC ;

     begin
     CLOCKTICK := times( TMS ) ;
     end ;


function CPUTICK : integer ;

     var

          TMS   : TMSREC  ;
          TRASH : integer ;

     begin
     TRASH := times( TMS ) ;
     with TMS do
          CPUTICK := UTIME + STIME + CUTIME + CSTIME ;
     end ;
```

```
$ page $


function DATESTRING : DATESTR ;

     var

          i        : integer ;
          POINTER  : DATEPTR ;
          SECONDS  : integer ;
          WORK     : DATESTR ;

     begin
     time ( SECONDS ) ;
     POINTER := ctime( SECONDS ) ;
     setstrlen ( WORK, DATELEN ) ;
     for i := 1 to DATELEN do
          WORK[i] := POINTER^[i] ;
     DATESTRING := WORK ;
     end ;


function NAMESTRING ( PAC : NAMEPAC ) : NAMESTR ;

     var

          i     : integer ;
          n     : integer ;
          WORK  : NAMESTR ;

     begin
     setstrlen( WORK, NAMELEN ) ;
     n := 0 ;
     for i := 1 to NAMELEN do
          if PAC[i] <> ' ' then
               begin
               n := n + 1 ;
               WORK[n] := PAC[i] ;
               end ;
     setstrlen ( WORK, n ) ;
     NAMESTRING := WORK ;
     end ;
```

```
$ page $


function UPPER_CASE( C : char ) : char ;

     var

           K : integer ;

     begin
     K := ord( C ) ;
     if K >= ord( 'a' ) then
          if K <= ord( 'z' ) then
                 C := chr( K + ord( 'A' ) - ord( 'a' ) ) ;
     UPPER_CASE := C ;
     end ;


function CAPWORD( WORD : WORDSTR ) : WORDSTR ;

     var

          i : integer ;
          n : integer ;

     begin
     n := strlen( WORD ) ;
     if n > 0 then
          for i := 1 to n do
               WORD[i] := UPPER_CASE( WORD[i] ) ;
     CAPWORD := WORD ;
     end ;
```

```
$ page $


function CHAR_INPUT( WMODE : CHWAITMODE ;
                     EMODE : CHECHOMODE ) : CHINPUTREC ;

    var

        K    : integer    ;
        WORK : CHINPUTREC ;

    begin
    setcbreak ;
    if EMODE = NOCHECHO then
        setnocecho ;
    if WMODE = NOCHWAIT then
        setnocwait ;
    K := kbdcharcode ;
    if ( WMODE = NOCHWAIT ) and ( K = -1 )

        then
        begin
        WORK.Q := NOTHING ;
        WORK.C := ' ' ;
        end

        else
        begin
        WORK.C := chr( K ) ;
        if WORK.C = NEWLINE

            then
            begin
            WORK.Q := ENDOFLINE ;
            WORK.C := ' ' ;
            end

            else
            WORK.Q := SOMETHING ;

        end ;

    setcwait ;
    setcecho ;
    setnocbreak ;
    CHAR_INPUT := WORK ;
    end ;
```

```
$ page $


function USER_DECIDES_TO( DO_THIS : PROMPTSTR ) : boolean ;

     var

          ANSWER     : string[ 3 ] ;
          OKAY       : boolean     ;
          WORK       : CHINPUTREC  ;


     begin
     SOUND_ALERT ;
     repeat
          SHOW ( DO_THIS+' ? ' ) ;
          WORK := CHAR_INPUT( CHWAIT, NOCHECHO ) ;
          OKAY := true ;
          if WORK.Q = ENDOFLINE

               then ANSWER := 'YES'

               else
               begin
               case WORK.C of

                    'Y','y':
                    ANSWER := 'YES' ;

                    'H','h','?':
                    begin
                    OKAY := false ;
                    SOUND_ALARM ;
                    end ;

                    otherwise
                    ANSWER :=  'NO' ;

                    end ; { case WORK.C }
               end ;

          CLEAR_LINE ;
          until OKAY ;
     SHOW ( DO_THIS+' ? ' ) ;
     SET_HALFBRIGHT_INVERSE_VIDEO ;
     SHOW ( ' '+ANSWER+' ' ) ;
     SET_NORMAL_VIDEO ;
     SHOWLN ( '' ) ;
     if ANSWER = 'NO'
          then USER_DECIDES_TO { DO_THIS } := false
          else USER_DECIDES_TO { DO_THIS } := true ;
     end ;
```

```
$ page $


function  WORD_INPUT( PROMPT   : PROMPTSTR   ;
                      DEFAULT : WORDSTR    ) : WORDSTR    ;

    begin
    WORD_INPUT := RJWORD_INPUT( PROMPT, DEFAULT, 0, WORDLEN ) ;
    end ;



function  RJWORD_INPUT( PROMPT   : PROMPTSTR   ;
                        DEFAULT : WORDSTR    ;
                        FIELD   : integer    ;
                        MAXLEN  : integer    ) : WORDSTR    ;

    var

        base            : integer ;
        current         : integer ;
        DELIM           : integer ;
        FLD             : integer ;
        K               : integer ;
        LBRACE          : integer ;
        LEN             : integer ;
        OPTION          : array [ 0..9 ] of WORDSTR ;
        RBRACE          : integer ;
        SELECT_MODE     : boolean ;
        VALUE           : WORDSTR ;
        VALUE_APPROVED  : boolean ;
        WERK            : CHINPUTREC ;
        WIRK            : LINESTR ;
        WORK            : LINESTR ;
```

```
$ page $

        begin { function RJWORD_INPUT }
        MAXLEN := IMIN( WORDLEN, MAXLEN ) ;
        WORK := strltrim( strrtrim( DEFAULT ) ) ;
        LEN := IMIN( MAXLEN, strlen( WORK ) ) ;
        DEFAULT := str( WORK, 1, LEN ) ;
        SELECT_MODE := false ;
        LBRACE := strpos( PROMPT, '{' ) ;
        if LBRACE > 0 then
             begin
             RBRACE := strpos( PROMPT, '}' ) ;
             if RBRACE > LBRACE then
                  begin
                  SELECT_MODE := true ;
                  base := 0 ;
                  LEN := RBRACE - LBRACE ;
                  WORK := str( PROMPT, LBRACE+1, LEN ) ;
                  repeat
                       DELIM := strpos( WORK, ',' ) ;
                       if DELIM = 0 then
                            DELIM := strpos( WORK, '}' ) ;
                       LEN := DELIM - 1 ;
                       WIRK := strltrim( strrtrim( str( WORK, 1, LEN ) ) ) ;
                       LEN := IMIN( MAXLEN, strlen( WIRK ) ) ;
                       OPTION[base] := str( WIRK, 1, LEN ) ;
                       strdelete( WORK, 1, DELIM ) ;
                       base := base + 1 ;
                       until strlen( WORK ) = 0 ;
                  current := base ;
                  repeat
                       current := current - 1 ;
                       until ( OPTION[current] = DEFAULT ) or ( current = 0 ) ;
                  DEFAULT := OPTION[current] ;
                  end ;
             end ;
        VALUE := DEFAULT ;
        SOUND_ALERT ;
        repeat
             SHOW ( PROMPT+' : ' ) ;
             SET_BRIGHTBLINK_INVERSE_VIDEO ;
             FLD := IMAX( FIELD, strlen( VALUE ) ) ;
             WORK := '' ;
             strwrite ( WORK,1,K,' ',VALUE:FLD,' ' ) ;
             SHOW ( WORK ) ;
             SET_NORMAL_VIDEO ;
```

```
$ page $

          if SELECT_MODE

               then
               begin
               WERK := CHAR_INPUT( CHWAIT, NOCHECHO ) ;
               if WERK.Q = ENDOFLINE

                    then VALUE_APPROVED := true

                    else
                    begin
                    VALUE_APPROVED := false ;
                    if WERK.C in [ '<','-' ]
                         then current := ( current - 1 ) mod base
                         else current := ( current + 1 ) mod base ;
                    VALUE := OPTION[current] ;
                    end ;

               CLEAR_LINE ;
               end

               else
               begin
               SHOWLN ( '' ) ;
               FETCHLN ( WORK ) ;
               MOVE_UP ;
               CLEAR_LINE ;
               MOVE_UP ;
               CLEAR_LINE ;
               if strlen( WORK ) = 0

                    then VALUE_APPROVED := true

                    else
                    begin
                    VALUE_APPROVED := false ;
                    WORK := strltrim( strrtrim( WORK ) ) ;
                    LEN := strlen( WORK ) ;
                    if LEN > MAXLEN

                         then
                         begin
                         LEN := MAXLEN ;
                         SOUND_ALARM ;
                         end

                         else SOUND_ALERT ;

                    VALUE := str( WORK, 1, LEN ) ;
                    end ;

               end ;

          until VALUE_APPROVED ;

     SHOW ( PROMPT+' : ' ) ;
     SET_HALFBRIGHT_INVERSE_VIDEO ;
     WORK := '' ;
```

```
    strwrite ( WORK,1,K,' ',VALUE:FLD,' ' ) ;
    SHOW ( WORK ) ;
    SET_NORMAL_VIDEO ;
    SHOWLN ( '' ) ;
    RJWORD_INPUT := VALUE ;
    end ; { function RJWORD_INPUT }
```

```
$ page $


function INTEGER_INPUT( PROMPT   : PROMPTSTR ;
                        DEFAULT  : integer   ;
                        FIELD    : integer   ) : integer ;

    var

        K                  : integer ;
        TRIAL_VALUE        : integer ;
        VALUE              : integer ;
        VALUE_APPROVED     : boolean ;
        WORK               : LINESTR ;

    begin
    VALUE := DEFAULT ;
    SOUND_ALERT ;
    repeat
        SHOW ( PROMPT+' : ' ) ;
        SET_BRIGHTBLINK_INVERSE_VIDEO ;
        WORK := '' ;
        strwrite ( WORK,1,K,' ',VALUE:FIELD,' ' ) ;
        SHOW ( WORK ) ;
        SET_NORMAL_VIDEO ;
        SHOWLN ( '' ) ;
        FETCHLN ( WORK ) ;
        MOVE_UP ;
        CLEAR_LINE ;
        MOVE_UP ;
        CLEAR_LINE ;
        if strlen( WORK ) = 0

            then
            VALUE_APPROVED := true

            else
            begin
            VALUE_APPROVED := false ;
            try                                    { set trap for possible error }
                strread ( WORK, 1, K, TRIAL_VALUE ) ;   { error here maybe }
                VALUE := TRIAL_VALUE ;
                SOUND_ALERT ;         { and jump around "recover" statement }
                recover SOUND_ALARM ;      { come here after error, if any }
            end ;

        until VALUE_APPROVED ;
    SHOW ( PROMPT+' : ' ) ;
    SET_HALFBRIGHT_INVERSE_VIDEO ;
    WORK := '' ;
    strwrite ( WORK,1,K,' ',VALUE:FIELD,' ' ) ;
    SHOW ( WORK ) ;
    SET_NORMAL_VIDEO ;
    SHOWLN ( '' ) ;
    INTEGER_INPUT := VALUE ;
    end ;
```

```
$ page $


function FIXED_INPUT( PROMPT  : PROMPTSTR ;
                      DEFAULT : longreal ;
                      FIELD   : integer  ;
                      PLACES  : integer  ) : longreal ;

     var

          K                : integer  ;
          TRIAL_VALUE      : longreal ;
          VALUE            : longreal ;
          VALUE_APPROVED   : boolean  ;
          WORK             : LINESTR  ;

     begin
     VALUE := DEFAULT ;
     SOUND_ALERT ;
     repeat
          WORK := '' ;
          strwrite ( WORK, 1, K, VALUE:FIELD:PLACES ) ;
          strread  ( WORK, 1, K, VALUE ) ;
          SHOW ( PROMPT+' : ' ) ;
          SET_BRIGHTBLINK_INVERSE_VIDEO ;
          SHOW ( ' '+WORK+' ' ) ;
          SET_NORMAL_VIDEO ;
          SHOWLN ( '' ) ;
          FETCHLN ( WORK ) ;
          MOVE_UP ;
          CLEAR_LINE ;
          MOVE_UP ;
          CLEAR_LINE ;
          if strlen( WORK ) = 0

               then
               VALUE_APPROVED := true

               else
               begin
               VALUE_APPROVED := false ;
               try                                { set trap for possible error }
                    strread ( WORK, 1, K, TRIAL_VALUE ) ;  { error here maybe }
                    VALUE := TRIAL_VALUE ;
                    SOUND_ALERT ;        { and jump around "recover" statement }
                    recover SOUND_ALARM ;      { come here after error, if any }
               end ;

          until VALUE_APPROVED ;
     SHOW ( PROMPT+' : ' ) ;
     SET_HALFBRIGHT_INVERSE_VIDEO ;
     WORK := '' ;
     strwrite ( WORK,1,K,' ',VALUE:FIELD:PLACES,' ' ) ;
     SHOW ( WORK ) ;
     SET_NORMAL_VIDEO ;
     SHOWLN ( '' ) ;
     FIXED_INPUT := VALUE ;
     end ;
```

```
end ;  { module UTILSPIF & File 'utilspif.I' }
```

```
$ page $  { begin File 'Utilvemq.I' }

{ Utility Software Unit for HP-9000 Series 200/300/500 Computers }

module UTILVEMQ ;  { Subject :  Vectors, Euler angles, matrices, quaternions }
                   { Domain  :  Universal                                     }

                   { NASA/JSC/MPAD/TRW      Sam Wilson }
                   { Updated  Thu Apr 10 23:22:11 1986 }

import

     UTILMATH ,
     UTILSPIF ;

export

     type

          EULARR =                               { a generic triad of Euler angles }
               array [ 1..3 ] of longreal ;      { representing three sequential   }
                                                 { rotations of a rigid body about }
                                                 { orthogonal axes fixed in the    }
                                                 { body itself:                    }
                                                 {      X axis = 1 = R = ROL       }
                                                 {      Y axis = 2 = P = PCH       }
                                                 {      Z axis = 3 = Y = YAW .     }
                                                 { The array is indexed according  }
                                                 { to the rotation number, NOT the }
                                                 { axis number.                    }

          EULPRY = EULARR ;        { Euler angles for a PCH/ROL/YAW sequence }
          EULPYR = EULARR ;        { Euler angles for a PCH/YAW/ROL sequence }
          EULRPR = EULARR ;        { Euler angles for a ROL/PCH/ROL sequence }
          EULYRY = EULARR ;        { Euler angles for a YAW/ROL/YAW sequence }

          VECTOR =                               { a row matrix of vector compon-  }
               array [ 1..3 ] of longreal ;      { ents in a right-handed Carte-   }
                                                 { sian coordinate system.  Index  }
                                                 { = coordinate axis number (phys- }
                                                 { ically) = matrix column number  }
                                                 { (mathematically).               }

          MAT3X3 =                               { a 3x3 matrix.  The element in   }
               array [ 1..3 ] of VECTOR ;        { the ith row and the jth column  }
                                                 { of matrix M can be addressed as }
                                                 { M[i,j].  The ith row (a vector) }
                                                 { can be addressed as M[i].       }

          DIAG3X3 =                              { the elements of a diagonal 3x3  }
               array [ 1..3 ] of longreal ;      { matrix M, stored in the order:  }
                                                 {     M[1,1] , M[2,2] , M[3,3] .  }

          SYMM3X3 =                              { the elements of a symmetric 3x3 }
               array [ 1..6 ] of longreal ;      { matrix M, stored in the order:  }
                                                 {     M[1,1] ,                    }
                                                 {     M[2,1] , M[2,2] ,           }
                                                 {     M[3,1] , M[3,2] , M[3,3] .  }
```

```
$ page $

          QUATERNION =                            { a record containing the sca- }
               record                             { lar and the vector parts of  }
               case boolean of                    { a quaternion.  The scalar    }
                    false : ( S : longreal ;      { part of the quaternion Q can }
                             V : VECTOR    ) ;    { be addressed as Q.S or Q.H,  }
                    true  : ( H : longreal ;      { and the vector part can be   }
                             I : longreal ;       { addressed as Q.V .  The in-  }
                             J : longreal ;       { dividual components of the   }
                             K : longreal ) ;     { the vector part can be ad-   }
               end ; { case and record }          { dressed as Q.V[1], Q.V[2],   }
                                                  { Q.V[3] or as Q.I, Q.J, Q.K . }

     const

          ZERVEC = VECTOR [ ZERO, ZERO, ZERO ] ;               { zero vector }
          XUNVEC = VECTOR [  ONE, ZERO, ZERO ] ;        { unit vector, axis 1 }
          YUNVEC = VECTOR [ ZERO,  ONE, ZERO ] ;        { unit vector, axis 2 }
          ZUNVEC = VECTOR [ ZERO, ZERO,  ONE ] ;        { unit vector, axis 3 }

          ZER3X3 = MAT3X3 [                                { 3x3 zero matrix }
               ZERVEC ,
               ZERVEC ,
               ZERVEC ] ;

          IDN3X3 = MAT3X3 [                            { 3x3 identity matrix }
               XUNVEC ,
               YUNVEC ,
               ZUNVEC ] ;


     function  EULDEG( E : EULARR ) : EULARR ;

               { EULDEG converts a triad of Euler angles from radian measure  }
               { to degrees.  It is normally used to convert internal values  }
               { to measurement units suitable for output.  This function is  }
               { unique in that angles output by every other routine in this  }
               { module are measured in radians.                              }


     function  EULRAD( E : EULARR ) : EULARR ;

               { This function converts a triad of Euler angles from degree   }
               { measure to radians.  It is normally used to convert external }
               { values supplied by the user to measurement units suitable    }
               { for internal computations.  EULRAD is unique in that the     }
               { input angles must of course be measured in degrees, whereas  }
               { angles input to every other routine defined in this module   }
               { must be measured in radians.                                 }


     function  DOTP( V , W : VECTOR ) : longreal ;

               { The value of this function is the dot (scalar) product of    }
               { the vectors V and W.                                         }
```

```
$ page $

      function  VMAG( V : VECTOR ) : longreal ;

                { The value of this function is the scalar magnitude of the    }
                { vector V.                                                     }


      function  SXV( S : longreal ; V : VECTOR   ) : VECTOR ;

                { The value of this function is the product of the scalar S    }
                { and the vector V.                                             }


      function  CRSP( V , W : VECTOR ) : VECTOR ;

                { The value of this function is the cross product of the vec-  }
                { tors V and W; i.e., CRSP(V,W) = V x W .                       }


      function  VDIF( V , W : VECTOR ) : VECTOR ;

                { The value of this function is the difference of the vectors  }
                { V and W; i.e., VDIF(V,W) = V - W .                            }


      function  VSUM( V , W : VECTOR ) : VECTOR ;

                { The value of this function is the sum of the vectors         }
                { V and W.                                                      }


      function  VXD( V : VECTOR ; D : DIAG3X3 ) : VECTOR ;

                { The value of this function is the product of the vector      }
                { (row matrix) V and the diagonal matrix D.  That is to say,   }
                { VXD(V,D) = V * M, where "*" represents the matrix multipli-  }
                { cation operator and M represents D in its full 3x3 form.     }


      function  VXM( V : VECTOR ; M : MAT3X3 ) : VECTOR ;

                { The value of this function is the product of the vector      }
                { (row matrix) V and the matrix M, i.e., VXM(V,M) = V * M,     }
                { where "*" represents the matrix multiplication operator.     }


      function  VXMT( V : VECTOR ; M : MAT3X3 ) : VECTOR ;

                { The value of this function is the product of the vector      }
                { (row matrix) V and the transpose of the matrix M, i.e.,      }
                { VXMT(V,M) = V * T, where "*" represents the matrix multi-    }
                { plication operator and T is the transpose of M.              }
```

```
$ page $

      function  MDIF( L , M : MAT3X3 ) : MAT3X3 ;

                { The value of this function is the difference of the matrices }
                { L and M; i.e., MDIF(L,M) = L - M .                           }


      function  MSUM( L , M : MAT3X3 ) : MAT3X3 ;

                { The value of this function is the sum of the matrices        }
                { L and M.                                                     }


      function  MXM( L , M : MAT3X3 ) : MAT3X3 ;

                { The value of this function is the product of the matrices    }
                { L and M; i.e., MXM(L,M) = L * M , where "*" represents the   }
                { matrix multiplication operator.                              }


      function  MXMT( L , M : MAT3X3 ) : MAT3X3 ;

                { The value of this function is the product of the matrix L    }
                { with the transpose of the matrix M; i.e., MXMT(L,M) = L * T, }
                { where "*" represents the matrix multiplication operator and  }
                { T is the transpose of M.                                     }

      function  MTXM( L , M : MAT3X3 ) : MAT3X3 ;

                { The value of this function is the product of the transpose   }
                { of the matrix L with the matrix  M; i.e., MTXM(L,M) = T * M, }
                { where "*" represents the matrix multiplication operator and  }
                { T is the transpose of L.                                     }


      function  MINV( M : MAT3X3 ) : MAT3X3 ;

                { The value of this function is the inverse of the matrix M,   }
                { which is computed by the INVERT_MATRIX procedure of module   }
                { UTILMATH.  INVERT_MATRIX will abort program execution with   }
                { an escapecode of 9901 if M is singular, and the value of     }
                { MINV will be undefined.  The reference to MINV should be     }
                { embedded in a "try/recover" construct if it is desired to    }
                { provide exception-handling code in the calling routine to    }
                { recover from such an eventuality.                            }
```

```
$ page $

      function  IMATQ( T : MAT3X3 ) : QUATERNION ;

                { Given the inverse (which is equal to the transpose T) of the }
                { orthogonal matrix that transforms vector components from one }
                { Cartesian coordinate system F to another Cartesian system G, }
                { this function computes a quaternion describing an eigenaxis  }
                { rotation which (assuming the two systems to have  a common   }
                { origin) would rotate the axes of system F into coincidence   }
                { with the axes of system G, and returns that quaternion to    }
                { the calling routine as the value of IMATQ.

                { This function is particularly useful for establishing the    }
                { angular relationship between a fixed reference system and    }
                { another system whose axes can be described in terms of unit  }
                { vectors in the reference frame.  For instance, if POS_J and  }
                { VEL_J are the position and velocity vectors of a satellite   }
                { in the Mean of 1950.0 geocentric inertial system J, the U    }
                { and W axes of the satellite-centered UVW coordinate system   }
                { are parallel to POS_J and POS_J x VEL_J, respectively.  The  }
                { angular orientation of the UVW system H is defined with re-  }
                { spect to J by the quaternion QJH, which can be determined by }
                { the following sequence of statements that also determine MHJ }
                { (the matrix that transforms vector components from system H  }
                { to system J).                                                }
                {                                                              }
                {       ANGMOM_J := CRSP( POS_J, VEL_J ) ;                     }
                {       MHJ[3] := SXV( ONE/VMAG( ANGMOM_J ), ANGMOM_J ) ;      }
                {       MHJ[1] := SXV( ONE/VMAG( POS_J ), POS_J ) ;            }
                {       MHJ[2] := CRSP( MHJ[3], MHJ[1] ) ;                     }
                {       QJH := IMATQ( MHJ ) ;                                  }
                {                                                              }

                { If the input matrix fails an orthogonality test based on the }
                { value of UNITOL (see module UTILMATH) then this routine will }
                { abort program execution with an escapecode of 9701, and the  }
                { value of IMATQ will be undefined.  The reference to IMATQ    }
                { should be embedded in a "try/recover" construct if it is     }
                { desired to provide exception-handling code in the calling    }
                { routine to recover from such an eventuality.                 }
```

```
$ page $

     function  PRYQ( PRY : EULPRY ) : QUATERNION ;

               { The value of this function is a quaternion that describes an }
               { eigenaxis rotation (i.e., a single rotation of a rigid body  }
               { about one suitably chosen axis) which has the same final     }
               { effect as the sequence of rotations defined by the input     }
               { values of the PCH/ROL/YAW Euler angles.                      }

               { The vector part of the output quaternion is referenced to    }
               { axes fixed in the rigid body, and it points in the direction }
               { of the eigenaxis rotation (i.e., in the direction that a     }
               { right-hand screw would travel linearly during a positive     }
               { rotation about the eigenaxis).  The scalar part is equal to  }
               { cos( A/2 ) and the magnitude of the vector part is equal to  }
               { sin( A/2 ), where A is the angle of rotation about the       }
               { eigenaxis.  Thus it can be seen that the sum of the squares  }
               { of the components of any quaternion used to define the ori-  }
               { entation of a rigid body must be equal to 1.0 (unity).  Such }
               { a quaternion is said to be "normalized" and is sometimes     }
               { referred to as a "versor", but more often it is called a     }
               { "unit quaternion".                                           }

               { When the "rigid body" is an imagined set of Cartesian axes   }
               { that is rotated about its origin from a state of coincidince }
               { with the axes of one fixed reference system into a state of  }
               { coincidence with the axes of another fixed reference system, }
               { it follows from the preceding paragraph that all components  }
               { of the quaternion --- including the components of its vector }
               { part --- have identical values in both of the fixed systems. }
               { Furthermore, the quaternion that describes the inverse rota- }
               { tion (from the second fixed system to the first) can be ob-  }
               { tained simply by reversing the sign on all components of the }
               { vector part --- or, alternatively, by changing the sign of   }
               { the scalar part and leaving the vector part unchanged.       }


     function  PYRQ( PYR : EULPYR ) : QUATERNION ;

               { This function is identical to PRYQ except that the Euler     }
               { rotation sequence is PCH/YAW/ROL instead of PCH/ROL/YAW.     }


     function  RPRQ( RPR : EULRPR ) : QUATERNION ;

               { This function is identical to PRYQ except that the Euler     }
               { rotation sequence is ROL/PCH/ROL instead of PCH/ROL/YAW.     }


     function  YRYQ( YRY : EULYRY ) : QUATERNION ;

               { This function is identical to PRYQ except that the Euler     }
               { rotation sequence is YAW/ROL/YAW instead of PCH/ROL/YAW.     }
```

```
$ page $

      function  QPRY( Q : QUATERNION ) : EULPRY ;

                { The value of this function is a triad of Euler angles for a  }
                { PCH/ROL/YAW rotation sequence having the same final effect   }
                { as that of the eigenaxis rotation which is defined by the    }
                { unit quaternion Q.                                            }


      function  QPYR( Q : QUATERNION ) : EULPYR ;

                { The value of this function is a triad of Euler angles for a  }
                { PCH/YAW/ROL rotation sequence having the same final effect   }
                { as that of the eigenaxis rotation which is defined by the    }
                { unit quaternion Q.                                            }


      function  QRPR( Q : QUATERNION ) : EULRPR ;

                { The value of this function is a triad of Euler angles for a  }
                { ROL/PCH/ROL rotation sequence having the same final effect   }
                { as that of the eigenaxis rotation which is defined by the    }
                { unit quaternion Q.                                            }


      function  QYRY( Q : QUATERNION ) : EULYRY ;

                { The value of this function is a triad of Euler angles for a  }
                { YAW/ROL/YAW rotation sequence having the same final effect   }
                { as that of the eigenaxis rotation which is defined by the    }
                { unit quaternion Q.                                            }


      function  QMAT( Q : QUATERNION ) : MAT3X3 ;

                { Given a unit quaternion Q describing an eigenaxis rotation    }
                { which (assuming coincidence of origins) will rotate the axes }
                { of Cartesian system F into coincidence with the axes of sys- }
                { tem G, this routine computes the matrix M that transforms    }
                { vector components from system F to system G, and returns it  }
                { to the calling routine as the value of the function QMAT.    }
                { For instance, if VEC_F is referenced to system F and VEC_G   }
                { is the physically identical vector referenced to system G,   }
                { then VEC_G = VXM( VEC_F, QMAT( QFG ) ), where QFG is a qua-  }
                { ternion that defines an eigenaxis rotation from F to G.      }

                { If the sum of the squares of the components of the input     }
                { quaternion differs from unity by more than UNITOL (see mod-  }
                { ule UTILMATH) then this routine will abort program execution }
                { with an escapecode of 9702, and the value of QMAT will be    }
                { undefined.  If it is desired to provide exception-handling   }
                { code in the calling routine to recover from such an eventu-  }
                { ality, the reference to QMAT should be embedded in a "try/   }
                { recover" construct.                                           }
```

```
$ page $


      function  QCXQ( P , Q : QUATERNION ) : QUATERNION ;

                { The value of this function is the quaternion product of the  }
                { conjugate of P with Q; i.e., QCXQ(P,Q) = C o Q , where "o"   }
                { (circle) represents the quaternion multiplication operator   }
                { and C is the conjugate of P (the quaternion that results     }
                { from reversing the sign on all components of the vector part }
                { of P).                                                       }


      function  QXQ( P , Q : QUATERNION ) : QUATERNION ;

                { The value of this function is the quaternion product of P    }
                { with Q; i.e., QXQ(P,Q) = P o Q, where "o" (circle) repre-    }
                { sents the quaternion multiplication operator.                }


      function  QXQC( P , Q : QUATERNION ) : QUATERNION ;

                { The value of this function is the quaternion product of P    }
                { with the conjugate of Q; i.e., QXQC(P,Q) = P o C, where "o"  }
                { (circle) represents the quaternion multiplication operator   }
                { and C is the conjugate of Q (the quaternion that results     }
                { from reversing the sign on all components of the vector part }
                { of Q).                                                       }


      function  UNIQUAT( Q : QUATERNION ) : QUATERNION ;

                { This routine normalizes the quaternion Q by computing the
                { sum of the squares of all its components, and then dividing  }
                { each of them by the square root of that sum.  The result is  }
                { returned to the calling routine as the value of UNIQUAT.     }
                { Normalization is required after each update of a (nominally) }
                { unit quaternion (such as one being used to define the ori-   }
                { entation of a rigid body) when it is being integrated numer- }
                { ically on a component-by-component basis (i.e., when the     }
                { components are being integrated as if they were separate     }
                { scalar variables).                                           }
```

```
$ page $

     function  ROT( V : VECTOR ; Q : QUATERNION ) : VECTOR ;

               { Given a vector V referenced to a Cartesian frame F, and a     }
               { unit quaternion Q defining the orientation of another frame   }
               { G with respect to F, this routine transforms V from F to G    }
               { and returns the transformed vector to the calling routine     }
               { as the value of ROT.  In terms of quaternion operations,      }
               { ROT(V,Q) = C o V o Q, where "o" (circle) represents the qua-  }
               { ternion multiplication operator, C represents the conjugate   }
               { of Q, and V is treated as a quaternion whose scalar part is   }
               { zero.  In terms of the equivalent matrix operations ,         }
               { ROT(V,Q) = VXM( V, QMAT( Q ) ), which represents the actual   }
               { method of computation used in this routine.                   }

               { If QFG is the name of the aforementioned quaternion in the    }
               { calling routine, and if VEC_F is the name of the vector       }
               { when it is referenced to frame F, then the statement          }
               {                                                               }
               {      VEC_G := ROT( VEC_F, QFG ) ;                             }
               {                                                               }
               { yields the value of the vector when referenced to frame G.    }
               { When more than one vector is to be transformed, e.g.,         }
               {                                                               }
               {      POS_G := ROT( POS_F, QFQ ) ;                             }
               {      VEL_G := ROT( VEL_F, QFG ) ;                             }
               {                                                               }
               { faster execution will be realized by the statement sequence   }
               {                                                               }
               {      MFG := QMAT( QFG ) ;                                     }
               {      POS_G := VXM( POS_F, MFG ) ;                             }
               {      VEL_G := VXM( VEL_F, MFG ) ;                             }
               {                                                               }
               { which minimizes the overhead associated with forming the      }
               { transformation matrix from the quaternion.                    }
```

```
$ page $

     function  IROT( V : VECTOR ; Q : QUATERNION ) : VECTOR ;

                 { IROT is the inverse of the ROT function.  That is, given a    }
                 { vector V referenced to a Cartesian frame G, and a unit        }
                 { quaternion Q defining the orientation of G with respect to    }
                 { another frame F, this routine transforms V from G to F        }
                 { and returns the transformed vector to the calling routine     }
                 { as the value of IROT.  In terms of quaternion operations,     }
                 { IROT(V,Q) = Q o V o C, where "o" (circle) represents the      }
                 { quaternion multiplication operator, C represents the conju-   }
                 { gate of Q, and V is treated as a quaternion whose scalar      }
                 { part is zero.  In terms of the equivalent matrix operations,  }
                 { IROT(V,Q) = VXMT( V, QMAT( Q ) ), which represents the ac-    }
                 { tual method of computation used in this routine.              }

                 { If QFG is the name of the aforementioned quaternion in the    }
                 { calling routine, and if VEC_G is the name of the vector       }
                 { when it is referenced to frame G, then the statement          }
                 {                                                               }
                 {      VEC_F := IROT( VEC_G, QFG ) ;                            }
                 {                                                               }
                 { yields the value of the vector when referenced to frame F.    }
                 { When more than one vector is to be transformed, e.g.,         }
                 {                                                               }
                 {      POS_F := IROT( POS_G, QFQ ) ;                            }
                 {      VEL_F := IROT( VEL_G, QFG ) ;                            }
                 {                                                               }
                 { faster execution will be realized by the statement sequence   }
                 {                                                               }
                 {      MFG := QMAT( QFG ) ;                                     }
                 {      POS_F := VXMT( POS_G, MFG ) ;                            }
                 {      VEL_F := VXMT( VEL_G, MFG ) ;                            }
                 {                                                               }
                 { which minimizes the overhead associated with forming the      }
                 { transformation matrix from the quaternion.                    }
```

```
$ page $


     procedure DIAGONALIZE ( S          : SYMM3X3  ;
                             TOLRATIO : longreal ;
                         var D          : DIAG3X3  ;
                         var M          : MAT3X3   ) ;

               { This procedure calls the DIAGONALIZE_SYMMATRIX procedure of  }
               { module UTILMATH, which uses the Jacobi method of iteration    }
               { to find an orthogonal matrix M that will (approximately)      }
               { transform an unknown diagonal matrix D into the symmetric     }
               { matrix S by use of the equation S = T * D * M, where "*" is   }
               { the matrix multiplication operator and T is the transpose of }
               { M.  Iteration ceases when a value of M is found such that     }
               { every off-diagonal element of  D' = M * S * T (which is an    }
               { approximation of D), has an absolute value no greater the     }
               { product of the input parameter TOLRATIO with the root-mean-   }
               { square magnitude of the diagonal elements of D.  After the    }
               { convergence test is satisfied, the diagonal elements of the   }
               { approximation D' are returned to the calling routine as the   }
               { components of the output variable D, along with the final     }
               { value of the transformation matrix M.                         }

               { If NERTENS_B is a symmetric 3x3 matrix representing a rigid- }
               { body inertia tensor referenced to an arbitrary body-fixed     }
               { Cartesian frame B, then the two statements                    }
               {                                                               }
               {      DIAGONALIZE ( NERTENS_B, 1.0L-6, NERTENS_P, MPB ) ;      }
               {      PYRBP := QPYR( IMATQ( MPB ) ) ;                          }
               {                                                               }
               { yield the principal moments of inertia (the three components }
               { of the diagonal matrix NERTENS_P) to a numerical  accuracy    }
               { of approximately one part in a million, plus the PCH/YAW/ROL }
               { Euler angles (PYRBP) that define the orientation of the       }
               { principal axes of inertia with respect to the B system.  MPB }
               { is a coordinate transformation matrix that can be used to     }
               { transform vectors from reference system P (whose axes coin-   }
               { cide with the principal axes of inertia) to system B.         }

               { If D' has not converged to the specified tolerance after 150 }
               { iterations, the values of M and D are NOT returned to the     }
               { calling routine,  and DIAGONALIZE_SYMMATRIX will abort pro-   }
               { gram execution with an escapecode of 9902.  The reference to }
               { DIAGONALIZE should be embedded in a "try/recover" construct   }
               { if it is desired to provide exception-handling code in the    }
               { calling routine to recover from such an eventuality.          }
```

```
$ page $

implement

     const

          HALF = 0.5L0 ;


function  EULDEG( E : EULARR ) : EULARR ;

     var

          j : integer ;

     begin
     for j := 1 to 3 do EULDEG[j] := ANGDEG( E[j] ) ;
     end ;


function  EULRAD( E : EULARR ) : EULARR ;

     var

          j : integer ;

     begin
     for j := 1 to 3 do EULRAD[j] := ANGRAD( E[j] ) ;
     end ;


function  DOTP( V , W : VECTOR ) : longreal ;

     begin
     DOTP := V[1] * W[1] + V[2] * W[2] + V[3] * W[3] ;
     end ;
```

```
$ page $


function  VMAG( V : VECTOR ) : longreal ;

     begin
     VMAG := sqrt( sqr( V[1] ) + sqr( V[2] ) + sqr( V[3] ) ) ;
     end ;


function  SXV( S : longreal ; V : VECTOR ) : VECTOR ;

     var

          j : integer ;

     begin
     for j := 1 to 3 do SXV[j] := S * V[j] ;
     end ;


function  CRSP( V , W : VECTOR ) : VECTOR ;

     begin
     CRSP[1] := V[2] * W[3] - W[2] * V[3] ;
     CRSP[2] := V[3] * W[1] - W[3] * V[1] ;
     CRSP[3] := V[1] * W[2] - W[1] * V[2] ;
     end ;
```

```
$ page $


function  VDIF( V , W : VECTOR ) : VECTOR ;

     var

          j : integer ;

     begin
     for j := 1 to 3 do VDIF[j] := V[j] - W[j] ;
     end ;


function  VSUM( V , W : VECTOR ) : VECTOR ;

     var

          j : integer ;

     begin
     for j := 1 to 3 do VSUM[j] := V[j] + W[j] ;
     end ;
```

```
$ page $


function  VXD( V : VECTOR ; D : DIAG3X3 ) : VECTOR ;

     var

          j : integer ;

     begin
     for j := 1 to 3 do VXD[j] := V[j] * D[j] ;
     end ;


function  VXM( V : VECTOR ; M : MAT3X3 ) : VECTOR ;

     var

          i : integer ;
          j : integer ;
          W : VECTOR  ;

     begin
     W := ZERVEC ;
     for j := 1 to 3 do
          for i := 1 to 3 do
               W[j] := W[j] + V[i] * M[i,j] ;
     VXM := W ;
     end ;


function  VXMT( V : VECTOR ; M : MAT3X3 ) : VECTOR ;

     var

          i : integer ;
          j : integer ;
          W : VECTOR  ;

     begin
     W := ZERVEC ;
     for j := 1 to 3 do
          for i := 1 to 3 do
               W[j] := W[j] + V[i] * M[j,i] ;
     VXMT := W ;
     end ;
```

```
$ page $


function  MDIF( L , M : MAT3X3 ) : MAT3X3 ;

     var

          i : integer ;
          j : integer ;
          N : MAT3X3  ;

     begin
     for i := 1 to 3 do
          for j := 1 to 3 do
               N[i,j] := L[i,j] - M[i,j] ;
     MDIF := N ;
     end ;


function  MSUM( L , M : MAT3X3 ) : MAT3X3 ;

     var

          i : integer ;
          j : integer ;
          N : MAT3X3  ;

     begin
     for i := 1 to 3 do
          for j := 1 to 3 do
               N[i,j] := L[i,j] + M[i,j] ;
     MSUM := N ;
     end ;
```

```
$ page $


function  MXM( L , M : MAT3X3 ) : MAT3X3 ;

     var

          i : integer ;
          j : integer ;
          k : integer ;
          N : MAT3X3  ;

     begin
     N := ZER3X3 ;
     for i := 1 to 3 do
          for j := 1 to 3 do
               for k := 1 to 3 do
                    N[i,j] := N[i,j] + L[i,k] * M[k,j] ;
     MXM := N ;
     end ;


function  MXMT( L , M : MAT3X3 ) : MAT3X3 ;

     var

          i : integer ;
          j : integer ;
          k : integer ;
          N : MAT3X3  ;

     begin
     N := ZER3X3 ;
     for i := 1 to 3 do
          for j := 1 to 3 do
               for k := 1 to 3 do
                    N[i,j] := N[i,j] + L[i,k] * M[j,k] ;
     MXMT := N ;
     end ;


function  MTXM( L , M : MAT3X3 ) : MAT3X3 ;

     var

          i : integer ;
          j : integer ;
          k : integer ;
          N : MAT3X3  ;

     begin
     N := ZER3X3 ;
     for i := 1 to 3 do
          for j:= 1 to 3 do
               for k := 1 to 3 do
                    N[i,j] := N[i,j] + L[k,i] * M[k,j] ;
     MTXM := N ;
     end ;
```

```
$ page $


function  MINV( M : MAT3X3 ) : MAT3X3 ;

     var

          N      : MAT3X3 ;

     begin
     INVERT_MATRIX ( M, 3, N ) ;
     MINV := N ;
     end ;


function  IMATQ( T : MAT3X3 ) : QUATERNION ;

     var

          F    : longreal ;
          i    : integer  ;
          j    : integer  ;
          R    : longreal ;
          SAVE : longreal ;
          TEST : longreal ;

     begin { function IMATQ }
     for i := 1 to 3 do
          for j := i to 3 do
               begin
               TEST := DOTP( T[i], T[j] ) ;
               if i = j then TEST := TEST - ONE ;
               if abs( TEST ) > UNITOL then
                    escape( 9701 ) ;                  { abort program execution }
               end ;
     TEST := ONE + T[1,1] + T[2,2] + T[3,3] ;
     if TEST >= ONE

          then
          begin { H comp }
          R        := sqrt( TEST ) ;
          IMATQ.H  := HALF * R ;
          F        := HALF / R ;
          IMATQ.I  := ( T[2,3] - T[3,2] ) * F ;
          IMATQ.J  := ( T[3,1] - T[1,3] ) * F ;
          IMATQ.K  := ( T[1,2] - T[2,1] ) * F ;
          end  { H comp }

          else
          begin { I test }
          SAVE := TWO - TEST ;
```

```
$ page $

          TEST := SAVE + TWO * T[1,1] ;
          if TEST >= ONE

               then
               begin { I comp }
               R        := sqrt( TEST ) ;
               IMATQ.I  := HALF * R ;
               F        := HALF / R ;
               IMATQ.J  := ( T[2,1] + T[1,2] ) * F ;
               IMATQ.K  := ( T[3,1] + T[1,3] ) * F ;
               IMATQ.H  := ( T[2,3] - T[3,2] ) * F ;
               end  { I comp }

               else
               begin { J test }
               TEST := SAVE + TWO * T[2,2] ;
               if TEST >= ONE

                    then
                    begin { J comp }
                    R        := sqrt( TEST ) ;
                    IMATQ.J  := HALF * R ;
                    F        := HALF / R ;
                    IMATQ.K  := ( T[3,2] + T[2,3] ) * F ;
                    IMATQ.H  := ( T[3,1] - T[1,3] ) * F ;
                    IMATQ.I  := ( T[1,2] + T[2,1] ) * F ;
                    end { J comp }

                    else
                    begin { K comp }
                    TEST     := SAVE + TWO * T[3,3] ;
                    R        := sqrt( TEST ) ;
                    IMATQ.K  := HALF * R ;
                    F        := HALF / R ;
                    IMATQ.H  := ( T[1,2] - T[2,1] ) * F ;
                    IMATQ.I  := ( T[1,3] + T[3,1] ) * F ;
                    IMATQ.J  := ( T[2,3] + T[3,2] ) * F ;
                    end ; { K comp }

               end ; { J test }

          end ; { I test }

     end ; { function IMATQ }
```

```
$ page $


function  PRYQ( PRY : EULPRY ) : QUATERNION ;

     var

          C      : array [ 1..3 ] of longreal    ;
          HAFANG : longreal ;
          m      : integer  ;
          S      : array [ 1..3 ] of longreal    ;

     begin
     for m := 1 to 3 do
          begin
          HAFANG := HALF * PRY[m] ;
          C[m]   := cos( HAFANG ) ;
          S[m]   := sin( HAFANG ) ;
          end ;
     PRYQ.H := C[1] * C[2] * C[3] + S[1] * S[2] * S[3] ;
     PRYQ.I := C[1] * S[2] * C[3] + S[1] * C[2] * S[3] ;
     PRYQ.J := S[1] * C[2] * C[3] - C[1] * S[2] * S[3] ;
     PRYQ.K := C[1] * C[2] * S[3] - S[1] * S[2] * C[3] ;
     end ;


function  PYRQ( PYR : EULPYR ) : QUATERNION ;

     var

          C      : array [ 1..3 ] of longreal    ;
          HAFANG : longreal ;
          m      : integer  ;
          S      : array [ 1..3 ] of longreal    ;

     begin
     for m := 1 to 3 do
          begin
          HAFANG := HALF * PYR[m] ;
          C[m]   := cos( HAFANG ) ;
          S[m]   := sin( HAFANG ) ;
          end ;
     PYRQ.H := C[1] * C[2] * C[3] - S[1] * S[2] * S[3] ;
     PYRQ.I := C[1] * C[2] * S[3] + S[1] * S[2] * C[3] ;
     PYRQ.J := S[1] * C[2] * C[3] + C[1] * S[2] * S[3] ;
     PYRQ.K := C[1] * S[2] * C[3] - S[1] * C[2] * S[3] ;
     end ;
```

```
$ page $


function  RPRQ( RPR : EULRPR ) : QUATERNION ;

     var

          C      : longreal ;
          HAFDIF : longreal ;
          HAFMID : longreal ;
          HAFSUM : longreal ;
          S      : longreal ;

     begin
     HAFMID := HALF * RPR[2] ;
     C      := cos( HAFMID ) ;
     S      := sin( HAFMID ) ;
     HAFDIF := HALF * ( RPR[1] - RPR[3] ) ;
     HAFSUM := HALF * ( RPR[1] + RPR[3] ) ;
     RPRQ.H := C * cos( HAFSUM ) ;
     RPRQ.I := C * sin( HAFSUM ) ;
     RPRQ.J := S * cos( HAFDIF ) ;
     RPRQ.K := S * sin( HAFDIF ) ;
     end ;


function  YRYQ( YRY : EULYRY ) : QUATERNION ;

     var

          C      : longreal ;
          HAFDIF : longreal ;
          HAFMID : longreal ;
          HAFSUM : longreal ;
          S      : longreal ;

     begin
     HAFMID := HALF * YRY[2] ;
     C      := cos( HAFMID ) ;
     S      := sin( HAFMID ) ;
     HAFDIF := HALF * ( YRY[1] - YRY[3] ) ;
     HAFSUM := HALF * ( YRY[1] + YRY[3] ) ;
     YRYQ.H := C * cos( HAFSUM ) ;
     YRYQ.I := S * cos( HAFDIF ) ;
     YRYQ.J := S * sin( HAFDIF ) ;
     YRYQ.K := C * sin( HAFSUM ) ;
     end ;
```

C-2

```
$ page $


function  QPRY( Q : QUATERNION ) : EULPRY ;

     var

          HAFDIF : longreal ;
          HAFSUM : longreal ;
          X      : longreal ;
          Y      : longreal ;
          Z      : longreal ;

     begin
     HAFDIF    := ATAN2( Q.J - Q.K , Q.H + Q.I ) ;
     HAFSUM    := ATAN2( Q.J + Q.K , Q.H - Q.I ) ;
     X         := sqr( Q.H ) - sqr( Q.I ) + sqr( Q.J ) - sqr( Q.K ) ;
     Y         := TWO * ( Q.I * Q.J + Q.H * Q.K ) ;
     Z         := TWO * ( Q.H * Q.I - Q.J * Q.K ) ;
     QPRY [1] := ANG2( HAFSUM + HAFDIF ) ;
     QPRY [2] := ATAN2( Z, sqrt( sqr( X ) + sqr( Y ) ) ) ;
     QPRY [3] := ANG2( HAFSUM - HAFDIF ) ;
     end ;


function  QPYR( Q : QUATERNION ) : EULPYR ;

     var

          HAFDIF : longreal ;
          HAFSUM : longreal ;
          X      : longreal ;
          Y      : longreal ;
          Z      : longreal ;

     begin
     HAFDIF    := ATAN2( Q.J - Q.I , Q.H - Q.K ) ;
     HAFSUM    := ATAN2( Q.J + Q.I , Q.H + Q.K ) ;
     X         := sqr( Q.H ) - sqr( Q.I ) + sqr( Q.J) - sqr( Q.K ) ;
     Y         := TWO * ( Q.J * Q.K - Q.H * Q.I ) ;
     Z         := TWO * ( Q.H * Q.K + Q.I * Q.J ) ;
     QPYR [1] := ANG2( HAFSUM + HAFDIF ) ;
     QPYR [2] := ATAN2( Z, sqrt( sqr( X ) + sqr( Y ) ) ) ;
     QPYR [3] := ANG2( HAFSUM - HAFDIF ) ;
     end ;
```

```
$ page $


function  QRPR( Q : QUATERNION ) : EULRPR ;

     var

          HAFDIF : longreal ;
          HAFSUM : longreal ;
          X      : longreal ;
          Y      : longreal ;
          Z      : longreal ;

     begin
     HAFDIF    := ATAN2( Q.K , Q.J ) ;
     HAFSUM    := ATAN2( Q.I , Q.H ) ;
     X         := sqr( Q.H ) + sqr( Q.I ) ;
     Y         := sqr( Q.J ) + sqr( Q.K ) ;
     QRPR [1] := ANG2( HAFSUM + HAFDIF ) ;
     QRPR [2] := TWO * ATAN2( sqrt( Y ) , sqrt( X ) ) ;
     QRPR [3] := ANG2( HAFSUM - HAFDIF ) ;
     end ;


function  QYRY( Q : QUATERNION ) : EULYRY ;

     var

          HAFDIF : longreal ;
          HAFSUM : longreal ;
          X      : longreal ;
          Y      : longreal ;
          Z      : longreal ;

     begin
     HAFDIF    := ATAN2( Q.J , Q.I ) ;
     HAFSUM    := ATAN2( Q.K , Q.H ) ;
     X         := sqr( Q.H ) + sqr( Q.K ) ;
     Y         := sqr( Q.I ) + sqr( Q.J ) ;
     QYRY [1] := ANG2( HAFSUM + HAFDIF ) ;
     QYRY [2] := TWO * ATAN2( sqrt( Y ) , sqrt( X ) ) ;
     QYRY [3] := ANG2( HAFSUM - HAFDIF ) ;
     end ;
```

```
$ page $


function  QMAT( Q : QUATERNION ) : MAT3X3 ;

     var

          A : longreal ;
          B : longreal ;
          C : longreal ;
          G : longreal ;
          H : longreal ;
          I : longreal ;
          J : longreal ;
          K : longreal ;
          X : longreal ;
          Y : longreal ;
          Z : longreal ;

     begin
     A := sqr( Q.I ) ;
     B := sqr( Q.J ) ;
     C := sqr( Q.K ) ;
     G := sqr( Q.H ) ;
     if abs( G + A + B + C - ONE ) > UNITOL then
          escape ( 9702 ) ;                          { abort program execution }
     H := G - A - B - C ;
     I := TWO * Q.H * Q.I ;
     J := TWO * Q.H * Q.J ;
     K := TWO * Q.H * Q.K ;
     X := TWO * Q.J * Q.K ;
     Y := TWO * Q.K * Q.I ;
     Z := TWO * Q.I * Q.J ;
     QMAT [1,1] := H + A + A ;
     QMAT [1,2] := Z - K ;
     QMAT [1,3] := Y + J ;
     QMAT [2,1] := Z + K ;
     QMAT [2,2] := H + B + B ;
     QMAT [2,3] := X - I ;
     QMAT [3,1] := Y - J ;
     QMAT [3,2] := X + I ;
     QMAT [3,3] := H + C + C ;
     end ;
```

```
$ page $

function  QCXQ( P , Q : QUATERNION ) : QUATERNION ;

     begin
     with P do
          begin
          QCXQ.H := H * Q.H + I * Q.I + J * Q.J + K * Q.K ;
          QCXQ.I := H * Q.I - I * Q.H - J * Q.K + K * Q.J ;
          QCXQ.J := H * Q.J - J * Q.H - K * Q.I + I * Q.K ;
          QCXQ.K := H * Q.K - K * Q.H - I * Q.J + J * Q.I ;
          end ;
     end ;

function  QXQ( P , Q : QUATERNION ) : QUATERNION ;

     begin
     with P do
          begin
          QXQ.H := H * Q.H - I * Q.I - J * Q.J - K * Q.K ;
          QXQ.I := H * Q.I + I * Q.H + J * Q.K - K * Q.J ;
          QXQ.J := H * Q.J + J * Q.H + K * Q.I - I * Q.K ;
          QXQ.K := H * Q.K + K * Q.H + I * Q.J - J * Q.I ;
          end ;
     end ;

function  QXQC( P , Q : QUATERNION ) : QUATERNION ;

     begin
     with P do
          begin
          QXQC.H := H * Q.H + I * Q.I + J * Q.J + K * Q.K ;
          QXQC.I := I * Q.H - H * Q.I - J * Q.K + K * Q.J ;
          QXQC.J := J * Q.H - H * Q.J - K * Q.I + I * Q.K ;
          QXQC.K := K * Q.H - H * Q.K - I * Q.J + J * Q.I ;
          end ;
     end ;

function  UNIQUAT( Q : QUATERNION ) : QUATERNION ;

     var

          F : longreal ;

     begin
     F   := ONE / sqrt( sqr( Q.S ) + DOTP( Q.V, Q.V ) ) ;
     UNIQUAT.S := F * Q.S ;
     UNIQUAT.V := SXV( F, Q.V ) ;
     end ;
```

```
$ page $


function  ROT( V : VECTOR ; Q : QUATERNION ) : VECTOR ;

     begin
     ROT := VXM( V, QMAT( Q ) ) ;
     end ;


function  IROT( V : VECTOR; Q : QUATERNION ) : VECTOR ;

     begin
     IROT := VXMT( V, QMAT( Q ) ) ;
     end ;


procedure DIAGONALIZE ( S        : SYMM3X3  ;
                        TOLRATIO : longreal ;
                    var D        : DIAG3X3  ;
                    var M        : MAT3X3   ) ;

     begin
     DIAGONALIZE_SYMMATRIX ( S, 3, TOLRATIO, D, M ) ;
     end ;

end ;  { module UTILVEMQ & File 'Utilvemq.I' }
```

```
$ page $  { begin File 'Utilstat.I' }

{ Utility Software Unit for HP-9000 Series 200/300/500 Computers }

module UTILSTAT ;  { Subject :  Statistics }
                   { Domain  :  Universal  }

                   { NASA/JSC/MPAD/TRW      Sam Wilson }
                   { Updated  Sat Apr 12 22:20:13 1986 }

import

     UTILMATH ,
     UTILSPIF ;

export

     type

          SIXVEC =                          { a six-vector (row matrix).          }
               array [1..6] of longreal ;   { Index = matrix column number.       }


          SIXPOPDEF =                       { an array defining a population      }
               array [1..21] of longreal ;  { of six-vectors whose components     }
                                            { are zero-mean random numbers        }
                                            { having correlated Gaussian          }
                    { (normal) distributions.  The  elements of the array, in }
                    { the order of storage, are:                              }
                    {                                                         }
                    {         S[1],                                           }
                    {       C[2,1],   S[2],                                   }
                    {       C[3,1], C[3,2],   S[3],                           }
                    {       C[4,1], C[4,2], C[4,3],   S[4],                   }
                    {       C[5,1], C[5,2], C[5,3], C[5,4],   S[5],           }
                    {       C[6,1], C[6,2], C[6,3], C[6,4], C[6,5],   S[6],   }
                    {                                                         }
                    { where S[j] is the standard deviation of the jth compo-  }
                    { nent and C[i,j] = C[j,i] is the coefficient of correla- }
                    { tion between components i and j.  The magnitudes of the }
                    { C[i,j] must be less than ONE (unity), and the S[j] must }
                    { be positive.  This array is related to the population   }
                    { covariance matrix COVAR through the equation            }
                    {                                                         }
                    {       COVAR[i,j] = S[i] * S[j] * C[i,j] ,               }
                    {                                                         }
                    { where C[j,j] = ONE by definition.  The 6x6 correlation  }
                    { matrix composed of the C[i,j] (including the ONE values }
                    { on the main diagonal) must be positive definite.        }


          TRIANG6X6 =                       { an array containing the nonzero   }
               array [1..21] of longreal ;  { elements of a 6x6 triangular      }
                                            { matrix (see TRIANGMAT type dec-   }
                                            { laration in module UTILMATH).     }
```

```
$ page $

     var

          RUNNUM   : integer ;  { Identification number for a simulation run     }
                                { in a Monte Carlo series.  If RUNNUM > 0,       }
                                { the assumed purpose of the run is to analyze }
                { the effects of random dispersions and errors and/or to gen-   }
                { erate Monte Carlo statistics, and all the pseudorandom num-   }
                { ber functions defined in this module (UNIFORM_RANDOM_SCALAR, }
                { GAUSSIAN_RANDOM_SCALAR, GAUSSIAN_RANDOM_SIXVECTOR) will have }
                { nonzero values.  If RUNNUM <= 0, the assumed purpose is to    }
                { analyze or to familiarize the simulator pilot with the nomi- }
                { nal situation, and the pseudorandom functions will have zero }
                { values.                                                       }


     function  UNIFORM_RANDOM_SCALAR( UNCERTAINTY : longreal ) : longreal ;

                { If RUNNUM > 0, the value of this function is a pseudorandom  }
                { number from a population which is uniformly distributed      }
                { over the interval between -UNCERTAINTY and +UNCERTAINTY.      }
                { If RUNNUM <=0, the value of this function is ZERO.            }


     function  GAUSSIAN_RANDOM_SCALAR( SIGMA : longreal ) : longreal ;

                { If RUNNUM > 0, the value of this function is a pseudorandom  }
                { number from a Gaussian (i.e., normally distributed) popula-  }
                { tion having a mean of zero and a standard deviation (square  }
                { root of variance) equal to  SIGMA.  If RUNNUM <= 0, the      }
                { value of this function is ZERO.                               }
```

```
$ page $

      function  SIXTUC_MATRIX( SIXPOP : SIXPOPDEF ) : TRIANG6X6 ;

                { The value of this function is an upper triangular 6x6 matrix }
                { that can be used to transform a six-vector of uncorrelated   }
                { zero-mean unit-variance Gaussian psedorandom numbers (U)     }
                { into a pseudorandom six-vector of correlated components (V)  }
                { from the population defined by SIXPOP.  The transformation   }
                { is defined by the equation                                   }
                {                                                              }
                {      V = U * SIXTUC_MATRIX( SIXPOP ) ,                       }
                {                                                              }
                { where "*" represents the matrix multiplication operator and  }
                { the components of U are GAUSSIAN_RANDOM_SCALAR function      }
                { values, viz:                                                 }
                {                                                              }
                {      U[j] = GAUSSIAN_RANDOM_SCALAR( ONE )                    }
                {                                                              }
                { for j = 1,2,....6 .                                          }

                { If any one of three possible error conditions are found to   }
                { exist in SIXPOP, the value of SIXTUC_MATRIX is undefined,    }
                { and program execution will be aborted.  The escapecode is    }
                { set equal to 9601 if one of the standard deviations (S[j])   }
                { is found to be zero or negative.  The escapecode is set to   }
                { 9602 if an off-diagonal coefficient of correlation (C[i,j])  }
                { is found to be greater than or equal to ONE.  If the corre-  }
                { lation matrix is found to be other than positive definite,   }
                { the escapecode is set equal to 9603.  The reference to       }
                { SIXTUC_MATRIX should be embedded in a "try/recover" con-     }
                { struct if it is desired to provide exception-handling code   }
                { to recover from such eventualities.                          }


      function  GAUSSIAN_RANDOM_SIXVECTOR( SIXTUC : TRIANG6X6 ) : SIXVEC ;

                { Given SIXTUC = SIXTUC_MATRIX( SIXPOP ), this routine will    }
                { compute a pseudorandom Gaussian six-vector from the popula-  }
                { tion defined by SIXPOP, and return it to the calling routine }
                { as the value of GAUSSIAN_RANDOM_SIXVECTOR.  If RUNNUM <= 0,  }
                { every component of GAUSSIAN_RANDOM_SIXVECTOR will have a     }
                { value of ZERO.                                               }
```

```
$ page $


implement

function  UNIFORM_RANDOM_SCALAR( UNCERTAINTY : longreal ) : longreal ;

     var

          R : longreal ;
          X : longreal ;

     begin
     R := maxint ;
     X := RANDOM_INTEGER / R ;
     if RUNNUM > 0
          then UNIFORM_RANDOM_SCALAR := ( TWO * X - ONE ) * UNCERTAINTY
          else UNIFORM_RANDOM_SCALAR := ( TWO * X - ONE ) * ZERO ;
     end ;


function  GAUSSIAN_RANDOM_SCALAR( SIGMA : longreal ) : longreal ;

     var

          G : longreal ;
          n : integer  ;
          X : longreal ;

     begin
     X := ZERO ;
     for n := 1 to 12 do
          X := X + RANDOM_INTEGER ;
     G := X / maxint - SIX ;
     if RUNNUM > 0
          then GAUSSIAN_RANDOM_SCALAR := G * SIGMA
          else GAUSSIAN_RANDOM_SCALAR := G * ZERO ;
     end ;
```

```
$ page $


function  SIXTUC_MATRIX( SIXPOP : SIXPOPDEF ) : TRIANG6X6 ;

          { Ref:  Subroutine SAMPLE in Program OMDAP, coded by Elric McHenry  }
          {       (modified by D. M. Braley), NASA/JSC/MPAD, ante April 1976  }

     const

          SIZE = 6 ;  { number of vector components }

     var

          i  : integer   ;
          ii : integer   ;
          ij : integer   ;
          j  : integer   ;
          jj : integer   ;
          k  : integer   ;
          M  : TRIANG6X6 ;
          X  : longreal  ;

     begin
     for i := 1 to SIZE do
          begin
          ii := TRIANG_INDEX( i, i ) ;
          if SIXPOP[ii] <= ZERO then
               escape ( 9601 ) ;                 { illegal standard deviation }
          X := ONE ;
          if i > 1 then
               for k := 1 to i-1 do
                    X := X - sqr( M[TRIANG_INDEX(k,i)] ) ;
          if X > ZERO
               then M[ii] := sqrt( X )
               else escape ( 9603 ) ; { correlation matrx is not pos definite }
          if i < SIZE then
               for j := i+1 to SIZE do
                    begin
                    ij := TRIANG_INDEX( i, j ) ;
                    if abs( SIXPOP[ij] ) < ONE
                         then X := SIXPOP[ij]
                         else escape ( 9602 ) ;   { illegal correlation coeff }
                    if i > 1 then
                         for k := 1 to i-1 do
                              X := X - M[TRIANG_INDEX(k,i)] *
                                       M[TRIANG_INDEX(k,j)] ;
                    M[ij] := X / M[ii] ;
                    end ;
          for j := i to SIZE do
               begin
               ij := TRIANG_INDEX( i, j ) ;
               jj := TRIANG_INDEX( j, j ) ;
               SIXTUC_MATRIX[ij] := M[ij] * SIXPOP[jj] ;
               end ;
          end ;
     end ;
```

```
$ page $


function  GAUSSIAN_RANDOM_SIXVECTOR( SIXTUC : TRIANG6X6 ) : SIXVEC ;

     var

          i  : integer ;
          ij : integer ;
          j  : integer ;
          k  : integer ;
          U  : SIXVEC  ;
          V  : SIXVEC  ;

     begin
     for j := 1 to 6 do
          U[j] := GAUSSIAN_RANDOM_SCALAR( ONE ) ;
     for j := 1 to 6 do
          begin
          V[j] := ZERO ;
          for i := 1 to j do
               V[j] := V[j] + U[i] * SIXTUC[TRIANG_INDEX(i,j)] ;
          end ;
     GAUSSIAN_RANDOM_SIXVECTOR := V ;
     end ;


end ;  { module UTILSTAT & File 'Utilstat.I' }
```

```
{ begin File 'UTILTEST.TEXT' }

{ Utility Software Unit for HP-9000 Model 216 with Pascal 3.0 Op Sys }

$ Sysprog On $
$ search 'UTILUNIT' $
$ Ref 60 $

program TEST_UTILITY_SOFTWARE_UNIT ( input, output ) ;

        { NASA/JSC/MPAD/TRW      Sam Wilson }
        { Updated  Sat Apr 12 22:35:51 1986 }

import

     UTILMATH ,
     UTILSPIF ,
     UTILVEMQ ,
     UTILSTAT ;

const

     D    = DIAG3X3 [   30.0L0,    10.0L0,    40.0L0 ] ;

     K    = MAT3X3  [
            VECTOR  [   25.0L0,     0.0L0,     0.0L0 ] ,
            VECTOR  [    0.0L0,    40.0L0,     0.0L0 ] ,
            VECTOR  [    0.0L0,     0.0L0,    55.0L0 ] ] ;

     L    = MAT3X3  [
            VECTOR  [      ONE,       TWO,    -THREE ] ,
            VECTOR  [     -TWO,      FIVE,       SIX ] ,
            VECTOR  [     FOUR,     THREE,     -FOUR ] ] ;

     M    = MAT3X3  [
            VECTOR  [      ONE,     THREE,      NINE ] ,
            VECTOR  [     FOUR,      FIVE,       SIX ] ,
            VECTOR  [    SEVEN,     EIGHT,       TWO ] ] ;

     PRY  = EULPRY  [ -145.0L0,    65.0L0, -170.0L0 ] ;
     PYR  = EULPYR  [   80.0L0,   -35.0L0,  120.0L0 ] ;
     RPR  = EULRPR  [   10.0L0,   -15.0L0,   20.0L0 ] ;
     YRY  = EULYRY  [   -5.0L0,    80.0L0,  -55.0L0 ] ;
     V    = VECTOR  [      TWO,      -SIX,    THREE ] ;
     W    = VECTOR  [     FOUR,      FIVE,     -ONE ] ;

type

     CLASSREC =                     { record of data pertaining to one class of }
          record                    { values for a continuous random variable   }
          VUB : longreal ;   { value of the upper bound for this class          }
          CDF : longreal ;   { cumulative distribution function for this class }
          end ; { record }

     CLASSARR =                                 { description of the distribution }
          array [ 0..10 ] of CLASSREC ;         { of a continuous random variable }

     DESCRIPSTR = string [ 22 ] ;   { descriptive text for an output quantity }
```

```
$ page $

const
                    {1234567890123456789012}
     NULLDESCRIP = '                      ' ;
     TENSPACES   = '          '             ;

var

     E        : DIAG3X3    ;
     H        : integer    ;
     N        : MAT3X3     ;
     OUTLINE  : LINESTR    ;
     P        : QUATERNION ;
     Q        : QUATERNION ;
     R        : QUATERNION ;
     S        : SYMM3X3    ;
     SAVTICK  : integer    ;
     TIME     : longreal   ;
     X        : VECTOR     ;

procedure PRINT_FIXED_SCALAR ( DESCRIP : DESCRIPSTR ; S : longreal ) ;forward;
procedure PRINT_FIXED_DIAG3X3( DESCRIP : DESCRIPSTR ; D : DIAG3X3  ) ;forward;
procedure PRINT_FIXED_EULARR ( DESCRIP : DESCRIPSTR ; E : EULARR   ) ;forward;
procedure PRINT_FIXED_VECTOR ( DESCRIP : DESCRIPSTR ; V : VECTOR   ) ;forward;
procedure PRINT_FIXED_MAT3X3 ( DESCRIP : DESCRIPSTR ; M : MAT3X3   ) ;forward;
procedure PRINT_FIXED_SIXVEC ( DESCRIP : DESCRIPSTR ; V : SIXVEC   ) ;forward;
procedure PRINT_FIXED_SIXPOP ( DESCRIP : DESCRIPSTR ; S : SIXPOPDEF) ;forward;

procedure PRINT_FLOAT_SCALAR ( DESCRIP : DESCRIPSTR ; S : longreal ) ;forward;
procedure PRINT_FLOAT_EULARR ( DESCRIP : DESCRIPSTR ; E : EULARR   ) ;forward;
procedure PRINT_FLOAT_VECTOR ( DESCRIP : DESCRIPSTR ; V : VECTOR   ) ;forward;
procedure PRINT_FLOAT_MAT3X3 ( DESCRIP : DESCRIPSTR ; M : MAT3X3   ) ;forward;
procedure PRINT_FLOAT_SIXPOP ( DESCRIP : DESCRIPSTR ; S : SIXPOPDEF) ;forward;

procedure TEST_UTILVEMQ_VECTOR_FUNCTIONS                             ;forward;
procedure TEST_UTILVEMQ_MATRIX_FUNCTIONS                             ;forward;
procedure TEST_UTILVEMQ_IMATQ_FUNCTION                               ;forward;
procedure TEST_UTILVEMQ_EULER_QUAT_FUNCTIONS                         ;forward;
procedure TEST_UTILVEMQ_QUAT_ROT_FUNCTIONS                           ;forward;
procedure TEST_UTILVEMQ_UNIQUAT_FUNCTION                             ;forward;
procedure TEST_UTILVEMQ_MATRIX_DIAGONALIZATION                       ;forward;
procedure TEST_UTILSTAT_UNIFORM_RANDOM_SCALAR_FUNCTION               ;forward;
procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SCALAR_FUNCTION              ;forward;
procedure TEST_UTILSTAT_SIXTUC_MATRIX_FUNCTION                       ;forward;
procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SIXVECTOR_FUNCTION           ;forward;

{ $ list off $ }   $ include 'Prtprocs.I.' $   { $ list on $ }
{ $ list off $ }   $ include 'Testmath.I.' $   { $ list on $ }
{ $ list off $ }   $ include 'Testspif.I.' $   { $ list on $ }
{ $ list off $ }   $ include 'Testvemq.I.' $   { $ list on $ }
{ $ list off $ }   $ include 'Teststat.I.' $   { $ list on $ }
```

```
$ page $

begin  { program TEST_UTILITY_SOFTWARE_UNIT }
INITIALIZE_IO ;
SAVTICK := CLOCKTICK ;
SHOWLN ( 'Test results will be saved in text file ''UTILTEST.R''' ) ;
rewrite ( LP, 'UTILTEST.R' ) ;
writeln ( LP, 'Utility Software Unit Test','Run':26,DATESTRING:27 ) ;
if USER_DECIDES_TO( 'Test UTILMATH module' ) then TEST_UTILMATH_MODULE ;
if USER_DECIDES_TO( 'Test UTILSPIF module' ) then TEST_UTILSPIF_MODULE ;
if USER_DECIDES_TO( 'Test UTILVEMQ module' ) then TEST_UTILVEMQ_MODULE ;
if USER_DECIDES_TO( 'Test UTILSTAT module' ) then TEST_UTILSTAT_MODULE ;
SHOWLN ( 'Tests completed' ) ;
SHOWLN ( ''  ) ;
OUTLINE := '' ;
strwrite ( OUTLINE,1,H,'Elapsed time = ',
                        ((CLOCKTICK-SAVTICK)/TICKSPERSEC):5:2,' seconds' ) ;
SHOWLN ( OUTLINE ) ;
SHOWLN ( '' ) ;
LOITER ( 500 ) ;
for H := 1 to 5 do writeln ( LP ) ;
writeln ( LP, 'Tests completed' ) ;
writeln ( LP ) ;
writeln ( LP, OUTLINE ) ;
writeln ( LP ) ;
close ( LP, 'SAVE' ) ;
CLEAN_UP_IO ;
end .  { program TEST_UTILITY_SOFTWARE_UNIT & File 'UTILTEST.TEXT' }
```

```
{ begin File 'utiltest.p' }

{ Utility Software Unit for HP-9000 Series 500 with HP-UX 5.0 Op Sys }

$ standard_level 'hp_modcal' $
$ search 'utilunit.o' $

program TEST_UTILITY_SOFTWARE_UNIT ( input, output ) ;

        { NASA/JSC/MPAD/TRW      Sam Wilson }
        { Updated  Sat Apr 12 22:52:11 1986 }

import

     UTILMATH ,
     UTILSPIF ,
     UTILVEMQ ,
     UTILSTAT ;

const

     D   = DIAG3X3 [   30.0L0,    10.0L0,    40.0L0 ] ;

     K   = MAT3X3  [
           VECTOR  [   25.0L0,     0.0L0,     0.0L0 ] ,
           VECTOR  [    0.0L0,    40.0L0,     0.0L0 ] ,
           VECTOR  [    0.0L0,     0.0L0,    55.0L0 ] ] ;

     L   = MAT3X3  [
           VECTOR  [      ONE,       TWO,    -THREE ] ,
           VECTOR  [     -TWO,      FIVE,       SIX ] ,
           VECTOR  [     FOUR,     THREE,     -FOUR ] ] ;

     M   = MAT3X3  [
           VECTOR  [      ONE,     THREE,      NINE ] ,
           VECTOR  [     FOUR,      FIVE,       SIX ] ,
           VECTOR  [    SEVEN,     EIGHT,       TWO ] ] ;

     PRY = EULPRY  [ -145.0L0,    65.0L0, -170.0L0 ] ;
     PYR = EULPYR  [   80.0L0,   -35.0L0,  120.0L0 ] ;
     RPR = EULRPR  [   10.0L0,   -15.0L0,   20.0L0 ] ;
     YRY = EULYRY  [   -5.0L0,    80.0L0,  -55.0L0 ] ;
     V   = VECTOR  [      TWO,      -SIX,    THREE ] ;
     W   = VECTOR  [     FOUR,      FIVE,     -ONE ] ;

type

     CLASSREC =                      { record of data pertaining to one class of }
          record                     { values for a continuous random variable   }
          VUB : longreal ;  { value of the upper bound for this class            }
          CDF : longreal ;  { cumulative distribution function for this class }
          end ; { record }

     CLASSARR =                                { description of the distribution }
          array [ 0..10 ] of CLASSREC ;       { of a continuous random variable }

     DESCRIPSTR = string [ 22 ] ;   { descriptive text for an output quantity }
```

```
$ page $

const
                   {1234567890123456789012}
     NULLDESCRIP = '                      ' ;
     TENSPACES   = '          '             ;

var

     E        : DIAG3X3    ;
     H        : integer    ;
     N        : MAT3X3     ;
     OUTLINE  : LINESTR    ;
     P        : QUATERNION ;
     Q        : QUATERNION ;
     R        : QUATERNION ;
     S        : SYMM3X3    ;
     SAVTICK  : integer    ;
     TIME     : longreal   ;
     X        : VECTOR     ;

procedure PRINT_FIXED_SCALAR ( DESCRIP : DESCRIPSTR ; S : longreal  ) ;forward;
procedure PRINT_FIXED_DIAG3X3( DESCRIP : DESCRIPSTR ; D : DIAG3X3   ) ;forward;
procedure PRINT_FIXED_EULARR ( DESCRIP : DESCRIPSTR ; E : EULARR    ) ;forward;
procedure PRINT_FIXED_VECTOR ( DESCRIP : DESCRIPSTR ; V : VECTOR    ) ;forward;
procedure PRINT_FIXED_MAT3X3 ( DESCRIP : DESCRIPSTR ; M : MAT3X3    ) ;forward;
procedure PRINT_FIXED_SIXVEC ( DESCRIP : DESCRIPSTR ; V : SIXVEC    ) ;forward;
procedure PRINT_FIXED_SIXPOP ( DESCRIP : DESCRIPSTR ; S : SIXPOPDEF ) ;forward;

procedure PRINT_FLOAT_SCALAR ( DESCRIP : DESCRIPSTR ; S : longreal  ) ;forward;
procedure PRINT_FLOAT_EULARR ( DESCRIP : DESCRIPSTR ; E : EULARR    ) ;forward;
procedure PRINT_FLOAT_VECTOR ( DESCRIP : DESCRIPSTR ; V : VECTOR    ) ;forward;
procedure PRINT_FLOAT_MAT3X3 ( DESCRIP : DESCRIPSTR ; M : MAT3X3    ) ;forward;
procedure PRINT_FLOAT_SIXPOP ( DESCRIP : DESCRIPSTR ; S : SIXPOPDEF ) ;forward;

procedure TEST_UTILVEMQ_VECTOR_FUNCTIONS                              ;forward;
procedure TEST_UTILVEMQ_MATRIX_FUNCTIONS                              ;forward;
procedure TEST_UTILVEMQ_IMATQ_FUNCTION                                ;forward;
procedure TEST_UTILVEMQ_EULER_QUAT_FUNCTIONS                          ;forward;
procedure TEST_UTILVEMQ_QUAT_ROT_FUNCTIONS                            ;forward;
procedure TEST_UTILVEMQ_UNIQUAT_FUNCTION                              ;forward;
procedure TEST_UTILVEMQ_MATRIX_DIAGONALIZATION                        ;forward;
procedure TEST_UTILSTAT_UNIFORM_RANDOM_SCALAR_FUNCTION                ;forward;
procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SCALAR_FUNCTION               ;forward;
procedure TEST_UTILSTAT_SIXTUC_MATRIX_FUNCTION                        ;forward;
procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SIXVECTOR_FUNCTION            ;forward;

{ $ list off $ }  $ include 'Prtprocs.I' $   { $ list on $ }
{ $ list off $ }  $ include 'Testmath.I' $   { $ list on $ }
{ $ list off $ }  $ include 'Testspif.I' $   { $ list on $ }
{ $ list off $ }  $ include 'Testvemq.I' $   { $ list on $ }
{ $ list off $ }  $ include 'Teststat.I' $   { $ list on $ }
```

```
$ page $

begin  { program TEST_UTILITY_SOFTWARE_UNIT }
INITIALIZE_IO ;
SAVTICK := CLOCKTICK ;
SHOWLN ( 'Test results will be saved in text file ''utiltest.R''' ) ;
rewrite ( LP, 'utiltest.R' ) ;
writeln ( LP, 'Utility Software Unit Test','Run':26,DATESTRING:27 ) ;
if USER_DECIDES_TO( 'Test UTILMATH module' ) then TEST_UTILMATH_MODULE ;
if USER_DECIDES_TO( 'Test UTILSPIF module' ) then TEST_UTILSPIF_MODULE ;
if USER_DECIDES_TO( 'Test UTILVEMQ module' ) then TEST_UTILVEMQ_MODULE ;
if USER_DECIDES_TO( 'Test UTILSTAT module' ) then TEST_UTILSTAT_MODULE ;
SHOWLN ( 'Tests completed' ) ;
SHOWLN ( ''  ) ;
OUTLINE := '' ;
strwrite ( OUTLINE,1,H,'Elapsed time = ',
                         ((CLOCKTICK-SAVTICK)/TICKSPERSEC):5:2,' seconds' ) ;
SHOWLN ( OUTLINE ) ;
SHOWLN ( '' ) ;
LOITER ( 500 ) ;
for H := 1 to 5 do writeln ( LP ) ;
writeln ( LP, 'Tests completed' ) ;
writeln ( LP ) ;
writeln ( LP, OUTLINE ) ;
writeln ( LP ) ;
close ( LP, 'SAVE' ) ;
CLEAN_UP_IO ;
end .  { program TEST_UTILITY_SOFTWARE_UNIT & File 'utiltest.p' }
```

```
$ page $  { begin File 'Prtprocs.I' }

{ Utility Software Unit for HP-9000 Series 200/300/500 Computers }

{ NASA/JSC/MPAD/TRW       Sam Wilson }
{ Updated  Thu Apr 10 23:33:60 1986 }


procedure PRINT_FIXED_SCALAR ( DESCRIP : DESCRIPSTR ; S : longreal ) ;

     begin
     writeln ( LP, DESCRIP:22,S:19:13 ) ;
     end ;


procedure PRINT_FIXED_DIAG3X3 ( DESCRIP : DESCRIPSTR ; D : DIAG3X3 ) ;

     var

          j : integer ;

     begin
     write   ( LP, DESCRIP:22 ) ;
     for j := 1 to 3 do write ( LP, D[j]:19:13 ) ;
     writeln ( LP ) ;
     end ;


procedure PRINT_FIXED_EULARR ( DESCRIP : DESCRIPSTR ; E : EULARR    ) ;

     var

          j : integer ;

     begin
     write   ( LP, DESCRIP:22 ) ;
     for j := 1 to 3 do write ( LP, E[j]:19:13 ) ;
     writeln ( LP ) ;
     end ;


procedure PRINT_FIXED_VECTOR ( DESCRIP : DESCRIPSTR ; V : VECTOR    ) ;

     var

          j : integer ;

     begin
     write   ( LP, DESCRIP:22 ) ;
     for j := 1 to 3 do write ( LP, V[j]:19:13 ) ;
     writeln ( LP ) ;
     end ;
```

```
$ page $


procedure PRINT_FIXED_MAT3X3 ( DESCRIP : DESCRIPSTR ; M : MAT3X3    ) ;

     var

          i : integer ;

     begin
     PRINT_FIXED_VECTOR ( DESCRIP, M[1] ) ;
     for i := 2 to 3 do PRINT_FIXED_VECTOR ( NULLDESCRIP, M[i] ) ;
     end ;


procedure PRINT_FIXED_SIXVEC ( DESCRIP : DESCRIPSTR ; V : SIXVEC   ) ;

     var

          j : integer ;
          K : integer ;
          L : LINESTR ;

     begin
     K := 1 ;
     L := '' ;
     strwrite ( L,K,K,DESCRIP:22 ) ;
     for j := 1 to 6 do strwrite ( L,K,K,V[j]:9:4 ) ;
     writeln ( LP, L:76 );
     end ;


procedure PRINT_FIXED_SIXPOP ( DESCRIP : DESCRIPSTR ; S : SIXPOPDEF ) ;

     var

          i : integer ;
          j : integer ;

     begin
     for i := 1 to 6 do
          begin
          if i = 1
               then write ( LP, DESCRIP:22 )
               else write ( LP, '':22 ) ;
          for j := 1 to i do
               write ( LP, S[TRIANG_INDEX(i,j)]:9:3 ) ;
          writeln ( LP ) ;
          end ;
     end ;
```

```
$ page $


procedure PRINT_FLOAT_SCALAR ( DESCRIP : DESCRIPSTR ; S : longreal ) ;

     begin
     writeln ( LP, DESCRIP:22,TENSPACES,S:9 ) ;
     end ;


procedure PRINT_FLOAT_EULARR ( DESCRIP : DESCRIPSTR ; E : EULARR    ) ;

     var

          j : integer ;

     begin
     write   ( LP, DESCRIP:22 ) ;
     for j := 1 to 3 do write ( LP, TENSPACES,E[j]:9 ) ;
     writeln ( LP ) ;
     end ;


procedure PRINT_FLOAT_VECTOR ( DESCRIP : DESCRIPSTR ; V : VECTOR    ) ;

     var

          j : integer ;

     begin
     write   ( LP, DESCRIP:22 ) ;
     for j := 1 to 3 do write ( LP, TENSPACES,V[j]:9 ) ;
     writeln ( LP ) ;
     end ;


procedure PRINT_FLOAT_MAT3X3 ( DESCRIP : DESCRIPSTR ; M : MAT3X3    ) ;

     var

          i : integer ;

     begin
     PRINT_FLOAT_VECTOR ( DESCRIP, M[1] ) ;
     for i := 2 to 3 do PRINT_FLOAT_VECTOR ( NULLDESCRIP, M[i] ) ;
     end ;
```

```
$ page $


procedure PRINT_FLOAT_SIXPOP ( DESCRIP : DESCRIPSTR ; S : SIXPOPDEF ) ;

     var

          i : integer ;
          j : integer ;

     begin
     for i := 1 to 6 do
          begin
          if i = 1
               then write ( LP, DESCRIP:22 )
               else write ( LP, '':22 ) ;
          for j := 1 to i do
               write ( LP, ' ',S[TRIANG_INDEX(i,j)]:8 ) ;
          writeln ( LP ) ;
          end ;
     end ;

{ end File 'Prtprocs.I' }
```

```
$ page $  { begin File 'Testmath.I' }

{ Utility Software Unit for HP-9000 Series 200/300/500 Computers }

procedure TEST_UTILMATH_MODULE ;

        { NASA/JSC/MPAD/TRW       Sam Wilson }
        { Updated  Thu Apr 10 23:37:17 1986 }

    const

        FLD = 45 ;

    var

        i : integer ;

    begin  { procedure TEST_UTILMATH_MODULE }
    for i := 1 to 2 do writeln ( LP ) ;
    writeln ( LP, 'TEST_UTILMATH_MODULE':49 ) ;
    for i := 1 to 5 do writeln ( LP ) ;

    writeln ( LP, 'INT( -2.3 ) = ':FLD,INT( -2.3 ):2 ) ;
    writeln ( LP, 'INT( -2.0 ) = ':FLD,INT( -2.0 ):2 ) ;
    writeln ( LP, 'INT( -0.3 ) = ':FLD,INT( -0.3 ):2 ) ;
    writeln ( LP, 'INT(  0.0 ) = ':FLD,INT(  0.0 ):2 ) ;
    writeln ( LP, 'INT(  1.3 ) = ':FLD,INT(  1.3 ):2 ) ;
    writeln ( LP ) ;

    writeln ( LP, 'FRAC( -2.3 ) = ':FLD,FRAC( -2.3 ):5:2 ) ;
    writeln ( LP, 'FRAC( -2.0 ) = ':FLD,FRAC( -2.0 ):5:2 ) ;
    writeln ( LP, 'FRAC( -0.3 ) = ':FLD,FRAC( -0.3 ):5:2 ) ;
    writeln ( LP, 'FRAC(  0.0 ) = ':FLD,FRAC(  0.0 ):5:2 ) ;
    writeln ( LP, 'FRAC(  1.3 ) = ':FLD,FRAC(  1.3 ):5:2 ) ;
    writeln ( LP ) ;

    writeln ( LP, 'RMOD( -2.8, -0.5 ) = ':FLD,RMOD( -2.8, -0.5 ):5:2 ) ;
    writeln ( LP, 'RMOD( -2.8,  0.0 ) = ':FLD,RMOD( -2.8,  0.0 ):5:2 ) ;
    writeln ( LP, 'RMOD( -2.8,  0.5 ) = ':FLD,RMOD( -2.8,  0.5 ):5:2 ) ;
    writeln ( LP ) ;
    writeln ( LP, 'RMOD(  2.8, -0.5 ) = ':FLD,RMOD(  2.8, -0.5 ):5:2 ) ;
    writeln ( LP, 'RMOD(  2.8,  0.0 ) = ':FLD,RMOD(  2.8,  0.0 ):5:2 ) ;
    writeln ( LP, 'RMOD(  2.8,  0.5 ) = ':FLD,RMOD(  2.8,  0.5 ):5:2 ) ;
    writeln ( LP ) ;

    writeln ( LP, 'RSIGN( -1.9 ) = ':FLD,RSIGN( -1.9 ):2 ) ;
    writeln ( LP, 'RSIGN(  0.0 ) = ':FLD,RSIGN(  0.0 ):2 ) ;
    writeln ( LP, 'RSIGN(  1.9 ) = ':FLD,RSIGN(  1.9 ):2 ) ;
    writeln ( LP ) ;

    writeln ( LP, 'ISIGN( -5 ) = ':FLD,ISIGN( -5 ):2 ) ;
    writeln ( LP, 'ISIGN(  0 ) = ':FLD,ISIGN(  0 ):2 ) ;
    writeln ( LP, 'ISIGN(  5 ) = ':FLD,ISIGN(  5 ):2 ) ;
    writeln ( LP ) ;
```

```
$ page $

     writeln ( LP, 'IMAX( -3, -4 ) = ':FLD,IMAX( -3, -4 ):2 ) ;
     writeln ( LP, 'IMAX(  3,  4 ) = ':FLD,IMAX(  3,  4 ):2 ) ;
     writeln ( LP ) ;

     writeln ( LP, 'IMIN( -3, -4 ) = ':FLD,IMIN( -3, -4 ):2 ) ;
     writeln ( LP, 'IMIN(  3,  4 ) = ':FLD,IMIN(  3,  4 ):2 ) ;
     writeln ( LP ) ;

     writeln ( LP, 'RMAX( -2.9, -3.9 ) = ':FLD,RMAX( -2.9, -3.9 ):5:2 ) ;
     writeln ( LP, 'RMAX(  2.9,  3.9 ) = ':FLD,RMAX(  2.9,  3.9 ):5:2 ) ;
     writeln ( LP ) ;

     writeln ( LP, 'RMIN( -2.9, -3.9 ) = ':FLD,RMIN( -2.9, -3.9 ):5:2 ) ;
     writeln ( LP, 'RMIN(  2.9,  3.9 ) = ':FLD,RMIN(  2.9,  3.9 ):5:2 ) ;

     START_NEW_PAGE ;
     for i := 1 to 9 do writeln ( LP ) ;

     writeln ( LP, 'ANGDEG(ONE) = ':FLD,ANGDEG(ONE):18:14 ) ;
     writeln ( LP, 'ANGRAD(ANGDEG(ONE))-ONE = ':FLD,
                   (ANGRAD(ANGDEG(ONE))-ONE):7 ) ;
     writeln ( LP ) ;

                   {12345678901234567890123456789012345}
     writeln ( LP, 'ANGDEG(ANG1(-THREE*TWOPI-HAFPI)) = ':FLD,
                    ANGDEG(ANG1(-THREE*TWOPI-HAFPI)):17:12 ) ;
     writeln ( LP, 'ANGDEG(ANG2( THREE*TWOPI-HAFPI)) = ':FLD,
                    ANGDEG(ANG2( THREE*TWOPI-HAFPI)):17:12 ) ;
     writeln ( LP ) ;


     writeln ( LP, 'ANGDEG(ATAN1(-SIX,SIX)) = ':FLD,
                    ANGDEG(ATAN1(-SIX,SIX)):17:12 ) ;
     writeln ( LP, 'ANGDEG(ATAN2(-SIX,SIX)) = ':FLD,
                    ANGDEG(ATAN2(-SIX,SIX)):17:12 ) ;
     writeln ( LP ) ;


     writeln ( LP, 'HMS( -36385.874L0 ) = ':FLD,HMS( -36385.874L0 ):12:6 ) ;
     writeln ( LP, 'HMS(  36385.874L0 ) = ':FLD,HMS(  36385.874L0 ):12:6 ) ;
     writeln ( LP ) ;

     writeln ( LP, 'SECS( -1006.25874L0 ) = ':FLD,
                    SECS( -1006.25874L0 ):13:6 ) ;
     writeln ( LP, 'SECS(  1006.25874L0 ) = ':FLD,
                    SECS(  1006.25874L0 ):13:6 ) ;
     writeln ( LP ) ;

     writeln ( LP, 'JULIAN_DAYNUM( 1980, 4, 2 ) = ':FLD,
                    JULIAN_DAYNUM( 1980, 4, 2 ):8 ) ;

     START_NEW_PAGE ;
     end ;  { procedure TEST_UTILMATH_MODULE }

{ end File 'Testmath.I' }
```

```
$ page $  { begin File 'Testspif.I' }

{ Utility Software Unit for HP-9000 Series 200/300/500 Computers }

procedure TEST_UTILSPIF_MODULE ;

          { NASA/JSC/MPAD/TRW      Sam Wilson }
          { Updated  Thu Apr 10 23:39:45 1986 }

     const

               {12345678901234567890123456789012345678901234S}
          P1 = 'Your name' ;
          P2 = 'Age................................(years)' ;
          P3 = 'Home town.................................' ;
          P4 = 'Direction from here.....{N,NE,E,SE,S,SW,W,NW}' ;
          P5 = 'Distance............................(miles)' ;

     var

          W1 : WORDSTR  ;
          I2 : integer  ;
          W3 : WORDSTR  ;
          W4 : WORDSTR  ;
          F5 : longreal ;

          I       : integer    ;
          n       : integer    ;
          OUTLINE : LINESTR    ;
          WORK    : CHINPUTREC ;
```

```
$ page $

     begin  { procedure TEST_UTILSPIF_MODULE }
     for n := 1 to 2 do writeln ( LP ) ;
     writeln ( LP, 'TEST_UTILSPIF_MODULE':49 ) ;
     for n := 1 to 5 do writeln ( LP ) ;

     SOUND_ALERT ;
     SHOW ( 'Waiting 2 seconds; if you press a key it will be echoed ' ) ;
     LOITER ( 2000 ) ;
     WORK := CHAR_INPUT( NOCHWAIT, CHECHO ) ;
     SHOWLN ( '' ) ;

     SOUND_ALERT ;
     SHOW ( 'Waiting indefinitely; press any key to continue "' ) ;
     OUTLINE := '' ;
     strwrite ( OUTLINE,1,I,CHAR_INPUT( CHWAIT, NOCHECHO ).C,'"' ) ;
     SHOWLN ( OUTLINE ) ;

     W1 := RJWORD_INPUT(  P1,       'Rumplestiltskin', 15, 15 ) ;
     I2 := INTEGER_INPUT( P2,                  maxint , 15     ) ;
     W3 := RJWORD_INPUT(  P3, 'Brno, Czechoslovakia', 15, 15 ) ;
     W4 := RJWORD_INPUT(  P4,                    'NE', 15, 15 ) ;
     F5 := FIXED_INPUT(   P5,         8299.11111877 , 15, 5  ) ;

     writeln ( LP, ('"'+W1+'"'):17 ) ;
     writeln ( LP, I2:16 ) ;
     writeln ( LP, ('"'+W3+'"'):17 ) ;
     writeln ( LP, ('"'+W4+'"'):17 ) ;
     writeln ( LP, F5:16:8 ) ;

     for n := 1 to 3 do writeln ( LP ) ;
     writeln ( LP, 'START_RANDOM_NUMBER_SEQUENCE ( 1 )' ) ;
     START_RANDOM_NUMBER_SEQUENCE ( 1 ) ;
     writeln ( LP ) ;
     writeln ( LP, 'Pseudorandom integers:' ) ;
     writeln ( LP ) ;
     for n := 1 to 10 do
          writeln ( LP, RANDOM_INTEGER:10 ) ;

     for n := 1 to 3 do writeln ( LP ) ;
     writeln ( LP, 'START_RANDOM_NUMBER_SEQUENCE ( 2147483646 )' ) ;
     START_RANDOM_NUMBER_SEQUENCE ( 2147483646 ) ;
     writeln ( LP ) ;
     writeln ( LP, 'Pseudorandom integers:' ) ;
     writeln ( LP ) ;
     for n := 1 to 10 do
          writeln ( LP, RANDOM_INTEGER:10 ) ;

     START_NEW_PAGE ;
     end ;  { procedure TEST_UTILSPIF_MODULE }

{ end File 'Testspif.I' }
```

```
$ page $  { begin File 'Testvemq.I' }

{ Utility Software Unit for HP-9000 Series 200/300/500 Computers }

procedure TEST_UTILVEMQ_MODULE ;

          { NASA/JSC/MPAD/TRW      Sam Wilson }
          { Updated  Thu Apr 10 23:47:14 1986 }

     var

          i : integer ;

     begin
     for i := 1 to 2 do writeln ( LP ) ;
     writeln ( LP, 'TEST_UTILVEMQ_MODULE':49 ) ;
     for i := 1 to 5 do writeln ( LP ) ;

     TEST_UTILVEMQ_VECTOR_FUNCTIONS ;
     TEST_UTILVEMQ_MATRIX_FUNCTIONS ;
     TEST_UTILVEMQ_IMATQ_FUNCTION ;
     TEST_UTILVEMQ_EULER_QUAT_FUNCTIONS ;
     TEST_UTILVEMQ_QUAT_ROT_FUNCTIONS ;
     TEST_UTILVEMQ_UNIQUAT_FUNCTION ;
     TEST_UTILVEMQ_MATRIX_DIAGONALIZATION ;

     end ;
```

```
$ page $

procedure TEST_UTILVEMQ_VECTOR_FUNCTIONS ;

     var

          i : integer ;
          j : integer ;

     begin
     PRINT_FIXED_VECTOR ( 'V =',V ) ;
     writeln ( LP ) ;
     PRINT_FIXED_VECTOR ( 'W =',W ) ;
     writeln ( LP ) ;
     PRINT_FIXED_SCALAR ( 'DOTP(V,W) =',DOTP(V,W) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_SCALAR ( 'VMAG(V) =',VMAG(V) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_VECTOR ( 'SXV(TWO,V) =',SXV(TWO,V) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_VECTOR ( 'CRSP(V,W) =',CRSP(V,W) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_VECTOR ( 'VDIF(V,W) =',VDIF(V,W) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_VECTOR ( 'VSUM(V,W) =',VSUM(V,W) ) ;
     for i := 1 to 5 do writeln ( LP ) ;

                    {1234567890123456789012}
     write   ( LP, '                      D  =' ) ;
     for j := 1 to 3 do write ( LP, D[j]:19:13 ) ;
     for i := 1 to 2 do writeln ( LP ) ;
     PRINT_FIXED_VECTOR ( 'VXD(V,D) =',VXD(V,D) ) ;
     for i := 1 to 5 do writeln ( LP ) ;

     PRINT_FIXED_MAT3X3 ( 'M =',M ) ;
     writeln ( LP ) ;
     PRINT_FIXED_VECTOR ( 'VXM(V,M) =',VXM(V,M) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_VECTOR ( 'VXMT(V,M) =',VXMT(V,M) ) ;

     START_NEW_PAGE ;
     end ;
```

```
$ page $

procedure TEST_UTILVEMQ_MATRIX_FUNCTIONS ;

     var

          i : integer ;

     begin
     for i := 1 to 9 do writeln ( LP ) ;

     PRINT_FIXED_MAT3X3 ( 'L =',L ) ;
     writeln ( LP ) ;
     PRINT_FIXED_MAT3X3 ( 'MDIF(L,M) =',MDIF(L,M) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_MAT3X3 ( 'MSUM(L,M) =',MSUM(L,M) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_MAT3X3 ( 'MXM(L,M) =',MXM(L,M) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_MAT3X3 ( 'MXMT(L,M) =',MXMT(L,M) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_MAT3X3 ( 'MTXM(L,M) =',MTXM(L,M) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_MAT3X3 ( 'MINV(M) =',MINV(M) ) ;
     writeln ( LP ) ;
     N := MXM(M,MINV(M)) ;
     PRINT_FIXED_MAT3X3 ( 'N = MXM(M,MINV(M)) =',N ) ;
     writeln ( LP ) ;
     PRINT_FLOAT_MAT3X3 ( 'MDIF(N,IDN3X3) =',MDIF(N,IDN3X3) ) ;

     START_NEW_PAGE ;
     end ;
```

```
$ page $

procedure TEST_UTILVEMQ_IMATQ_FUNCTION ;

     var

          i : integer ;

     begin
     for i := 1 to 9 do writeln ( LP ) ;

     X := CRSP(W,V) ;
     PRINT_FIXED_VECTOR ( 'X = CRSP(W,V) =',X ) ;
     writeln ( LP ) ;
                    {1234567890123456789012}
     writeln ( LP, '                      N[3] = SXV( ONE/VMAG( X ), X )' ) ;
     writeln ( LP, '                      N[1] = SXV( ONE/VMAG( W ), W )' ) ;
     writeln ( LP, '                      N[2] = CRSP( N[3], N[1] )' ) ;
     writeln ( LP ) ;
     N[3] := SXV( ONE/VMAG( X ), X )  ;
     N[1] := SXV( ONE/VMAG( W ), W )  ;
     N[2] := CRSP( N[3], N[1] )  ;
     PRINT_FIXED_MAT3X3 ( 'N =',N ) ;
     writeln ( LP ) ;
                    {1234567890123456789012}
     writeln ( LP, '                          Q = IMATQ( N )' ) ;
     writeln ( LP ) ;
     Q := IMATQ( N ) ;
     PRINT_FIXED_SCALAR ( 'Q.S =',Q.S ) ;
     PRINT_FIXED_VECTOR ( 'Q.V =',Q.V ) ;
     writeln ( LP ) ;
                    {1234567890123456789012}
     writeln ( LP, '                          P = QCXQ( Q, Q )' ) ;
     writeln ( LP ) ;
     P := QCXQ( Q, Q )  ;
     PRINT_FIXED_SCALAR ( 'P.S =',P.S ) ;
     PRINT_FIXED_VECTOR ( 'P.V =',P.V ) ;
     writeln ( LP ) ;
     PRINT_FLOAT_SCALAR ( 'ONE - P.S =',ONE-P.S ) ;
     PRINT_FLOAT_VECTOR ( ' P.V =',P.V ) ;
     writeln ( LP ) ;
     N := MXMT( N, N ) ;
     PRINT_FIXED_MAT3X3 ( 'N = MXMT( N, N ) =',N ) ;
     writeln ( LP ) ;
     PRINT_FLOAT_MAT3X3 ( 'MDIF(N,IDN3X3) =',MDIF(N,IDN3X3) ) ;

     START_NEW_PAGE ;
     end ;
```

```
$ page $

procedure TEST_UTILVEMQ_EULER_QUAT_FUNCTIONS ;

     var

          i : integer ;

     begin
     for i := 1 to 9 do writeln ( LP ) ;

     PRINT_FIXED_EULARR ( 'PRY =',PRY ) ;
     writeln ( LP ) ;
     PRINT_FIXED_EULARR ( 'RPR =',RPR ) ;
     writeln ( LP ) ;
                        {1234567890123456789012}
     writeln ( LP, '                        P = PRYQ( EULRAD( PRY ) )' ) ;
     writeln ( LP, '                        Q = RPRQ( EULRAD( RPR ) )' ) ;
     writeln ( LP ) ;
     P := PRYQ(EULRAD(PRY)) ;
     Q := RPRQ(EULRAD(RPR)) ;
     PRINT_FIXED_SCALAR ( 'P.S =',P.S ) ;
     PRINT_FIXED_VECTOR ( 'P.V =',P.V ) ;
     writeln ( LP ) ;
     PRINT_FIXED_SCALAR ( 'Q.S =',Q.S ) ;
     PRINT_FIXED_VECTOR ( 'Q.V =',Q.V ) ;
     writeln ( LP ) ;
     PRINT_FIXED_EULARR ( 'EULDEG( QPRY( P ) ) =',EULDEG( QPRY( P ) ) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_EULARR ( 'EULDEG( QRPR( Q ) ) =',EULDEG( QRPR( Q ) ) ) ;
     for i := 1 to 5 do writeln ( LP ) ;

     PRINT_FIXED_EULARR ( 'PYR =',PYR ) ;
     writeln ( LP ) ;
     PRINT_FIXED_EULARR ( 'YRY =',YRY ) ;
     writeln ( LP ) ;
                        {1234567890123456789012}
     writeln ( LP, '                        P = PYRQ( EULRAD( PYR ) )' ) ;
     writeln ( LP, '                        Q = YRYQ( EULRAD( YRY ) )' ) ;
     writeln ( LP ) ;
     P := PYRQ(EULRAD(PYR)) ;
     Q := YRYQ(EULRAD(YRY)) ;
     PRINT_FIXED_SCALAR ( 'P.S =',P.S ) ;
     PRINT_FIXED_VECTOR ( 'P.V =',P.V ) ;
     writeln ( LP ) ;
     PRINT_FIXED_SCALAR ( 'Q.S =',Q.S ) ;
     PRINT_FIXED_VECTOR ( 'Q.V =',Q.V ) ;
     writeln ( LP ) ;
     PRINT_FIXED_EULARR ( 'EULDEG( QPYR( P ) ) =',EULDEG( QPYR( P ) ) ) ;
     writeln ( LP ) ;
     PRINT_FIXED_EULARR ( 'EULDEG( QYRY( Q ) ) =',EULDEG( QYRY( Q ) ) ) ;

     START_NEW_PAGE ;
     end ;
```

```
$ page $

procedure TEST_UTILVEMQ_UNIQUAT_FUNCTION ;

     var

          i : integer ;

     begin
     for i := 1 to 8 do writeln ( LP ) ;

     P.S := PI * R.S ;
     P.V := SXV( PI, R.V ) ;
     Q := UNIQUAT( P ) ;
                    {1234567890123456789012}
     writeln ( LP, '                       P.S = PI * R.S' ) ;
     writeln ( LP, '                       P.V = SXV( PI, R.V )' ) ;
     writeln ( LP ) ;
     PRINT_FIXED_SCALAR ( 'P.S =',P.S ) ;
     PRINT_FIXED_VECTOR ( 'P.V =',P.V ) ;
     writeln ( LP ) ;
                    {1234567890123456789012}
     writeln ( LP, '                          Q = UNIQUAT( P )' ) ;
     writeln ( LP ) ;
     PRINT_FIXED_SCALAR ( 'Q.S =',Q.S ) ;
     PRINT_FIXED_VECTOR ( 'Q.V =',Q.V ) ;
     writeln ( LP ) ;
     R := QXQC( Q, Q ) ;
                    {1234567890123456789012}
     writeln ( LP, '                          R = QXQC( Q, Q )' ) ;
     writeln ( LP ) ;
     PRINT_FLOAT_SCALAR ( 'ONE - R.S =',ONE-R.S ) ;
     PRINT_FLOAT_VECTOR ( 'R.V =',R.V ) ;
     for i := 1 to 5 do writeln ( LP ) ;

     end ;
```

PRECEDING PAGE BLANK NOT FILMED

```
$ page $

procedure TEST_UTILVEMQ_MATRIX_DIAGONALIZATION ;

     var

          i          : integer ;
          MPB        : MAT3X3  ;
          NERTENS_B  : MAT3X3  ;
          NERTENS_P  : MAT3X3  ;
          RPRBP      : EULRPR  ;
          RPRPB      : EULRPR  ;

     begin
     RPRPB := RPR ;
     PRINT_FIXED_EULARR ( 'RPRPB =',RPRPB ) ;
     writeln ( LP ) ;
     NERTENS_P := K ;
     PRINT_FIXED_MAT3X3 ( 'NERTENS_P =',NERTENS_P ) ;
     writeln ( LP ) ;
     MPB := QMAT( RPRQ( EULRAD( RPRPB ) ) ) ;
     NERTENS_B := MTXM( MPB, MXM( NERTENS_P, MPB ) ) ;
                    {1234567890123456789012}
     writeln ( LP, '                    MPB = QMAT(RPRQ(EULRAD( RPRPB )))' ) ;
     writeln ( LP, '              NERTENS_B = MTXM( MPB,MXM( NERTENS_P, MPB))' ) ;
     writeln ( LP ) ;
     PRINT_FIXED_MAT3X3 ( 'NERTENS_B =',NERTENS_B ) ;
     writeln ( LP ) ;
     S[1] :=   NERTENS_B[1,1] ;
     S[2] := ( NERTENS_B[2,1] + NERTENS_B[1,2] ) / TWO ;
     S[3] :=   NERTENS_B[2,2] ;
     S[4] := ( NERTENS_B[3,1] + NERTENS_B[1,3] ) / TWO ;
     S[5] := ( NERTENS_B[3,2] + NERTENS_B[2,3] ) / TWO ;
     S[6] :=   NERTENS_B[3,3] ;
     writeln ( LP, 'S =  NERTENS_B[1,1],                                 ':70 ) ;
     writeln ( LP, '     NERTENS_B[2,1], NERTENS_B[2,2],                 ':70 ) ;
     writeln ( LP, '     NERTENS_B[3,1], NERTENS_B[3,2], NERTENS_B[3,3]':70 ) ;
     writeln ( LP ) ;
     DIAGONALIZE ( S, 1.0L-8, E, N ) ;
                    {1234567890123456789012}
     writeln ( LP, '                     DIAGONALIZE ( S, 1.0L-8, E, N )' ) ;
     writeln ( LP ) ;
     PRINT_FIXED_DIAG3X3 ( 'E =',E ) ;
     writeln ( LP ) ;
     RPRBP := EULDEG( QRPR( IMATQ( N ) ) ) ;
                    {1234567890123456789012}
     writeln ( LP, '                  RPRBP = EULDEG( QRPR( IMATQ( N ) ) )' ) ;
     writeln ( LP ) ;
     PRINT_FIXED_EULARR ( 'RPRBP =',RPRBP ) ;

     START_NEW_PAGE ;
     end ;

{ end File 'Testvemq.I' }
```

```
$ page $  { begin File 'Teststat.I' }

{ Utility Software Unit for HP-9000 Series 200/300/500 Computers }

procedure TEST_UTILSTAT_MODULE ;

          { NASA/JSC/MPAD/TRW      Sam Wilson }
          { Updated  Sat Apr 12 22:39:45 1986 }

     var

          i : integer ;

     begin
     for i := 1 to 2 do writeln ( LP ) ;
     writeln ( LP, 'TEST_UTILSTAT_MODULE':49 ) ;
     for i := 1 to 5 do writeln ( LP ) ;

     RUNNUM := 1 ;
     START_RANDOM_NUMBER_SEQUENCE ( 454387819 ) ;
     TEST_UTILSTAT_UNIFORM_RANDOM_SCALAR_FUNCTION ;
     TEST_UTILSTAT_GAUSSIAN_RANDOM_SCALAR_FUNCTION ;
     TEST_UTILSTAT_SIXTUC_MATRIX_FUNCTION ;
     TEST_UTILSTAT_GAUSSIAN_RANDOM_SIXVECTOR_FUNCTION ;

     end ;
```

```
$ page $

procedure TEST_UTILSTAT_UNIFORM_RANDOM_SCALAR_FUNCTION ;

     const

          NUMVALS   =  1000 ;
          UNCERT    =  FIVE ;

          CLASS = CLASSARR [
               CLASSREC [ VUB : -5.0L0 , CDF : 0.0L0 ] ,
               CLASSREC [ VUB : -4.0L0 , CDF : 0.1L0 ] ,
               CLASSREC [ VUB : -3.0L0 , CDF : 0.2L0 ] ,
               CLASSREC [ VUB : -2.0L0 , CDF : 0.3L0 ] ,
               CLASSREC [ VUB : -1.0L0 , CDF : 0.4L0 ] ,
               CLASSREC [ VUB :  0.0L0 , CDF : 0.5L0 ] ,
               CLASSREC [ VUB :  1.0L0 , CDF : 0.6L0 ] ,
               CLASSREC [ VUB :  2.0L0 , CDF : 0.7L0 ] ,
               CLASSREC [ VUB :  3.0L0 , CDF : 0.8L0 ] ,
               CLASSREC [ VUB :  4.0L0 , CDF : 0.9L0 ] ,
               CLASSREC [ VUB :  5.0L0 , CDF : 1.0L0 ] ] ;

     var

          A       : integer  ;
          CPUTIME : longreal ;
          E       : integer  ;
          h       : integer  ;
          k       : integer  ;
          L       : LINESTR  ;
          M       : integer  ;
          n       : integer  ;
          DISTRIB : array [ 0..10 ] of integer ;
          X       : longreal ;
          SAVTICK : integer  ;
          R       : longreal ;
```

```
$ page $

begin  { procedure TEST_UTILSTAT_UNIFORM_RANDOM_SCALAR_FUNCTION }
SHOWLN ( '' ) ;
for k := 0 to 10 do DISTRIB[k] := 0 ;
SAVTICK := CPUTICK ;
for n := 1 to NUMVALS do
     begin
     if ( n mod 100 ) = 0 then
          begin
          L := '' ;
          strwrite ( L,1,M,'UNIFORM_RANDOM_SCALAR ',n:5 ) ;
          CLEAR_LINE ;
          SHOW ( L ) ;
          end ;
     X := UNIFORM_RANDOM_SCALAR( UNCERT ) ;
     for k := 0 to 10 do
          if X <= CLASS[k].VUB then
               DISTRIB[k] := DISTRIB[k] + 1 ;
     end ;
SHOWLN ( '' ) ;
CPUTIME := ( CPUTICK - SAVTICK ) / TICKSPERSEC ;
writeln ( LP, 'Test UNIFORM_RANDOM_SCALAR Function':57 ) ;
for h := 1 to 8 do writeln ( LP ) ;
writeln ( LP, ' CUMULATIVE DISTRIBUTION OF ':40,NUMVALS:4,
              ' PSEUDORANDOM NUMBERS' ) ;
writeln ( LP, 'FROM A':42 ) ;
writeln ( LP, 'UNIFORMLY DISTRIBUTED POPULATION':55 ) ;
writeln ( LP, 'HAVING ZERO MEAN':47 ) ;
writeln ( LP, '( UNCERTAINTY = ':44,UNCERT:4:1,' )' ) ;
writeln ( LP ) ;
writeln ( LP, 'CPU Time = ':40,CPUTIME:6:2,' sec' ) ;
writeln ( LP ) ;
writeln ( LP, '   CLASS         ACTUAL        EXPECTED       ACTUAL /':64 );
writeln ( LP, 'UPPER BOUND   DISTRIBUTION  DISTRIBUTION    EXPECTED':64 );
writeln ( LP ) ;
for k := 0 to 10 do
     begin
     A := DISTRIB[k] ;
     E := round( NUMVALS * CLASS[k].CDF ) ;
     if A = E

          then
          R := ONE

          else
          if E = 0
               then R := 99999.9999
               else R := A / E ;

     writeln ( LP, CLASS[k].VUB:20:1,A:14,E:14,R:15:4 ) ;
     end ;
START_NEW_PAGE ;
end ;  { procedure TEST_UTILSTAT_UNIFORM_RANDOM_SCALAR_FUNCTION }
```

```
$ page $

procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SCALAR_FUNCTION ;

     const

          NUMVALS  =  1000 ;
          SIGMA    =   TEN ;

          CLASS = CLASSARR [
               CLASSREC [ VUB : -25.0L0 , CDF : 0.00621L0 ] ,
               CLASSREC [ VUB : -20.0L0 , CDF : 0.02275L0 ] ,
               CLASSREC [ VUB : -15.0L0 , CDF : 0.06681L0 ] ,
               CLASSREC [ VUB : -10.0L0 , CDF : 0.15866L0 ] ,
               CLASSREC [ VUB :  -5.0L0 , CDF : 0.30854L0 ] ,
               CLASSREC [ VUB :   0.0L0 , CDF : 0.50000L0 ] ,
               CLASSREC [ VUB :   5.0L0 , CDF : 0.69146L0 ] ,
               CLASSREC [ VUB :  10.0L0 , CDF : 0.84134L0 ] ,
               CLASSREC [ VUB :  15.0L0 , CDF : 0.93319L0 ] ,
               CLASSREC [ VUB :  20.0L0 , CDF : 0.97725L0 ] ,
               CLASSREC [ VUB :  25.0L0 , CDF : 0.99379L0 ] ] ;

     var

          A       : integer  ;
          CPUTIME : longreal ;
          E       : integer  ;
          h       : integer  ;
          k       : integer  ;
          L       : LINESTR  ;
          M       : integer  ;
          n       : integer  ;
          DISTRIB : array [ 0..10 ] of integer ;
          X       : longreal ;
          SAVTICK : integer  ;
          R       : longreal ;
```

```
$ page $

     begin  { procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SCALAR_FUNCTION }
     SHOWLN ( '' ) ;
     for k := 0 to 10 do DISTRIB[k] := 0 ;
     SAVTICK := CPUTICK ;
     for n := 1 to NUMVALS do
          begin
          if ( n mod 100 ) = 0 then
               begin
               L := '' ;
               strwrite ( L,1,M,'GAUSSIAN_RANDOM_SCALAR ',n:5 ) ;
               CLEAR_LINE ;
               SHOW ( L ) ;
               end ;
          X := GAUSSIAN_RANDOM_SCALAR( SIGMA ) ;
          for k := 0 to 10 do
               if X <= CLASS[k].VUB then
                    DISTRIB[k] := DISTRIB[k] + 1 ;
          end ;
     SHOWLN ( '' ) ;
     CPUTIME := ( CPUTICK - SAVTICK ) / TICKSPERSEC ;
     for h := 1 to 7 do writeln ( LP ) ;
     writeln ( LP, 'Test GAUSSIAN_RANDOM_SCALAR Function':57 ) ;
     for h := 1 to 8 do writeln ( LP ) ;
     writeln ( LP, ' CUMULATIVE DISTRIBUTION OF ':40,NUMVALS:4,
                   ' PSEUDORANDOM NUMBERS' ) ;
     writeln ( LP, 'FROM A':42 ) ;
     writeln ( LP, 'NORMALLY DISTRIBUTED POPULATION':55 ) ;
     writeln ( LP, 'HAVING ZERO MEAN':47 ) ;
     writeln ( LP, '( SIGMA = ':41,SIGMA:4:1,' )' ) ;
     writeln ( LP ) ;
     writeln ( LP, 'CPU Time = ':40,CPUTIME:6:2,' sec' ) ;
     writeln ( LP ) ;
     writeln ( LP, '   CLASS         ACTUAL       EXPECTED      ACTUAL /':64 );
     writeln ( LP, 'UPPER BOUND   DISTRIBUTION  DISTRIBUTION    EXPECTED':64 );
     writeln ( LP ) ;
     for k := 0 to 10 do
          begin
          A := DISTRIB[k] ;
          E := round( NUMVALS * CLASS[k].CDF ) ;
          if A = E

               then
               R := ONE

               else
               if E = 0
                    then R := 99999.9999
                    else R := A / E ;

          writeln ( LP, CLASS[k].VUB:20:1,A:14,E:14,R:15:4 ) ;
          end ;
     START_NEW_PAGE ;
     end ;  { procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SCALAR_FUNCTION }
```

```
$ page $


procedure TEST_UTILSTAT_SIXTUC_MATRIX_FUNCTION ;

     const

          SIXPOPIN = SIXPOPDEF [

  1.00000L0,
  0.48650L0,   1.00000L0,
  0.53460L0,   0.69872L0,   1.00000L0,
 -0.47810L0,  -0.99618L0,  -0.70584L0,   1.00000L0,
 -0.19910L0,  -0.66165L0,  -0.62706L0,   0.66691L0,   1.00000L0,
 -0.46049L0,  -0.77813L0,  -0.87129L0,   0.78915L0,   0.72009L0,   1.00000L0 ];

     var

          h          : integer   ;
          i          : integer   ;
          ij         : integer   ;
          j          : integer   ;
          k          : integer   ;
          ki         : integer   ;
          kj         : integer   ;
          SIXTUC     : TRIANG6X6 ;
          SIXPOPOUT  : SIXPOPDEF ;
```

```
$ page $

     begin  { procedure TEST_UTILSTAT_SIXTUC_MATRIX_FUNCTION }
     for h := 1 to 6 do writeln ( LP ) ;
     writeln ( LP, 'Test SIXTUC_MATRIX Function':53 ) ;
     for h := 1 to 7 do writeln ( LP ) ;
     PRINT_FIXED_SIXPOP ( 'SIXPOPIN  =',SIXPOPIN ) ;
     for h := 1 to 2 do writeln ( LP ) ;
     try
          SIXTUC := SIXTUC_MATRIX( SIXPOPIN ) ;
          writeln ( LP, ' SIXPOPOUT =':22,
                        ' T * M ,  where M is SIXTUC_MATRIX( SIXPOPIN )' ) ;
          writeln ( LP, '':22,
                        '          and T is the transpose of M.' ) ;
          for h := 1 to 2 do writeln ( LP ) ;
          for i := 1 to 6 do
               for j := 1 to i do
                    begin
                    ij := TRIANG_INDEX( i, j ) ;
                    SIXPOPOUT[ij] := ZERO ;
                    for k := 1 to j do
                         begin
                         ki := TRIANG_INDEX( k, i ) ;
                         kj := TRIANG_INDEX( k, j ) ;
                         SIXPOPOUT[ij] := SIXPOPOUT[ij] + SIXTUC[ki] *
                                                          SIXTUC[kj] ;
                         end ;
                    end ;
          PRINT_FIXED_SIXPOP ( 'SIXPOPOUT =',SIXPOPOUT ) ;
          for h := 1 to 2 do writeln ( LP ) ;
          for i := 1 to 6 do
               for j := 1 to i do
                    begin
                    ij := TRIANG_INDEX( i, j ) ;
                    SIXPOPOUT[ij] := abs( SIXPOPOUT[ij] - SIXPOPIN[ij] ) ;
                    end ;
          PRINT_FLOAT_SIXPOP ( '|SIXPOPOUT-SIXPOPIN| =',SIXPOPOUT ) ;
          for h := 1 to 2 do writeln ( LP ) ;

          recover
          writeln ( LP, 'escapecode = ',escapecode:1 ) ;

     START_NEW_PAGE ;
     end ;  { procedure TEST_UTILSTAT_SIXTUC_MATRIX_FUNCTION }
```

```
$ page $

procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SIXVECTOR_FUNCTION ;

     const

          MAXPOPS =     3 ;
          NUMVALS =  1000 ;

     type

          POPIDNUM     = 1..MAXPOPS ;
          POPIDTEXTARR = array [ POPIDNUM ] of LINESTR   ;
          SIXPOPDEFARR = array [ POPIDNUM ] of SIXPOPDEF ;

     const

          POPIDTEXT = POPIDTEXTARR [

LINESTR [ 'ASTP  / Apollo STDN 2-Station Estimation Error                  ' ],
LINESTR [ 'PAIDS / Long-Range Stationkeep without TIC ; 0.5 deg Deadbands' ],
LINESTR [ 'PAIDS / Long-Range Stationkeep with TIC ; 3.0 deg Deadbands    ' ]];

          SIXPOPIN = SIXPOPDEFARR [

SIXPOPDEF [  { ASTP  / Apollo STDN 2-Station Estimation Error }

 75.30095L0,
  0.48650L0, 179.90586L0,
  0.53460L0,   0.69872L0,  79.85549L0,
 -0.47810L0,  -0.99618L0,  -0.70584L0,   0.18992L0,
 -0.19910L0,  -0.66165L0,  -0.62706L0,   0.66691L0,   0.11363L0,
 -0.46049L0,  -0.77813L0,  -0.87129L0,   0.78915L0,   0.72009L0,   0.41689L0 ],

SIXPOPDEF [  { PAIDS / Long-Range Stationkeep without TIC ; 0.5 deg Deadbands }

 10.10000L0,
  0.37596L0,  40.30000L0,
 -0.19199L0,   0.16483L0,  11.90000L0,
 -0.40541L0,  -0.65226L0,  -0.08834L0,   0.06300L0,
  0.55319L0,  -0.05193L0,  -0.35843L0,   0.06776L0,   0.07300L0,
 -0.34093L0,   0.27385L0,   0.18710L0,  -0.20776L0,  -0.36069L0,   0.02600L0 ],

SIXPOPDEF [  { PAIDS / Long-Range Stationkeep with TIC ; 3.0 deg Deadbands }

  6.90000L0,
  0.43070L0,  27.80000L0,
  0.30076L0,   0.09788L0,   6.90000L0,
 -0.46684L0,  -0.36040L0,   0.33187L0,   0.04200L0,
  0.51904L0,   0.51485L0,   0.32888L0,   0.10399L0,   0.06300L0,
 -0.07188L0,   0.05597L0,   0.36320L0,   0.06798L0,  -0.01471L0,   0.02400L0]];
```

```
$ page $

     var

          COVAROUT    : array [ 1..6, 1..6 ] of longreal ;
          CPUTIME     : longreal  ;
          h           : integer   ;
          i           : integer   ;
          ii          : integer   ;
          ij          : integer   ;
          j           : integer   ;
          jj          : integer   ;
          k           : integer   ;
          L           : LINESTR   ;
          M           : integer   ;
          population  : POPIDNUM  ;
          SIXTUC      : TRIANG6X6 ;
          n           : integer   ;
          SAVTICK     : integer   ;
          SIXPOPOUT   : SIXPOPDEF ;
          V           : SIXVEC    ;

     begin  { procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SIXVECTOR_FUNCTION }
     for population := 1 to 3 do
          begin
          SHOWLN ( '' ) ;
          for h := 1 to 6 do writeln ( LP ) ;
          writeln ( LP, 'Test GAUSSIAN_RANDOM_SIXVECTOR Function':59 ) ;
          for h := 1 to 3 do writeln ( LP ) ;
          writeln ( LP, TENSPACES,' ',POPIDTEXT[population] ) ;
          for h := 1 to 3 do writeln ( LP ) ;
          PRINT_FIXED_SIXPOP ( 'SIXPOPIN  =',SIXPOPIN[population] ) ;
          for h := 1 to 2 do writeln ( LP ) ;
          for i := 1 to 6 do
               for j := 1 to 6 do
                    COVAROUT[i,j] := ZERO ;
          SAVTICK := CPUTICK ;
```

```
$ page $

          try
               SIXTUC := SIXTUC_MATRIX( SIXPOPIN[population] ) ;
               for n := 1 to NUMVALS do
                    begin
                    if ( n mod 100 ) = 0 then
                         begin
                         L := '' ;
                         strwrite ( L,1,M,'GAUSSIAN_RANDOM_SIXVECTOR ',n:5 ) ;
                         CLEAR_LINE ;
                         SHOW ( L ) ;
                         end ;
                    V := GAUSSIAN_RANDOM_SIXVECTOR( SIXTUC ) ;
                    for i := 1 to 6 do
                         for j := 1 to 6 do
                              COVAROUT[i,j] := COVAROUT[i,j] + V[i] * V[j] ;
                    end ;
               SHOWLN ( '' ) ;
               CPUTIME := ( CPUTICK - SAVTICK ) / TICKSPERSEC ;
               writeln ( LP, ' SIXPOPOUT =':22,
                             ' statistical summary of ',NUMVALS:4,
                             ' pseudorandom' ) ;
               writeln ( LP, '':22,' six-vectors from population defined by',
                             ' SIXPOPIN' ) ;
               for h := 1 to 2 do writeln ( LP ) ;
               writeln ( LP, 'CPU time = ':23,CPUTIME:4:2,' sec' ) ;
               for h := 1 to 2 do writeln ( LP ) ;
               for i := 1 to 6 do
                    for j := 1 to 6 do
                         COVAROUT[i,j] := COVAROUT[i,j] / NUMVALS ;
               for j := 1 to 6 do
                    begin
                    jj := TRIANG_INDEX( j, j ) ;
                    SIXPOPOUT[jj] := sqrt( COVAROUT[j,j] ) ;
                    end ;
               for j := 1 to 5 do
                    begin
                    jj := TRIANG_INDEX( j, j ) ;
                    for i := j+1 to 6 do
                         begin
                         ii := TRIANG_INDEX( i, i ) ;
                         ij := TRIANG_INDEX( i, j ) ;
                         SIXPOPOUT[ij] := COVAROUT[i,j] /
                                          (SIXPOPOUT[ii]*SIXPOPOUT[jj]) ;
                         end ;
                    end ;
               PRINT_FIXED_SIXPOP ( 'SIXPOPOUT =',SIXPOPOUT ) ;

               recover
               writeln ( LP, 'escapecode = ',escapecode:1 ) ;
          START_NEW_PAGE ;
          end ;
     end ;  { procedure TEST_UTILSTAT_GAUSSIAN_RANDOM_SIXVECTOR_FUNCTION }

{ end File 'Teststat.I' }
```

TEST_UTILMATH_MODULE

```
            INT( -2.3 ) = -3
            INT( -2.0 ) = -2
            INT( -0.3 ) = -1
            INT(  0.0 ) =  0
            INT(  1.3 ) =  1

           FRAC( -2.3 ) =  0.70
           FRAC( -2.0 ) =  0.00
           FRAC( -0.3 ) =  0.70
           FRAC(  0.0 ) =  0.00
           FRAC(  1.3 ) =  0.30

     RMOD( -2.8, -0.5 ) = -0.30
     RMOD( -2.8,  0.0 ) =  0.00
     RMOD( -2.8,  0.5 ) =  0.20

     RMOD(  2.8, -0.5 ) = -0.20
     RMOD(  2.8,  0.0 ) =  0.00
     RMOD(  2.8,  0.5 ) =  0.30

          RSIGN( -1.9 ) = -1
          RSIGN(  0.0 ) =  1
          RSIGN(  1.9 ) =  1

            ISIGN( -5 ) = -1
            ISIGN(  0 ) =  1
            ISIGN(  5 ) =  1

         IMAX( -3, -4 ) = -3
         IMAX(  3,  4 ) =  4

         IMIN( -3, -4 ) = -4
         IMIN(  3,  4 ) =  3

     RMAX( -2.9, -3.9 ) = -2.90
     RMAX(  2.9,  3.9 ) =  3.90

     RMIN( -2.9, -3.9 ) = -3.90
     RMIN(  2.9,  3.9 ) =  2.90
```

```
                  ANGDEG(ONE) =   57.2957795130823
       ANGRAD(ANGDEG(ONE))-ONE = -2.2E-016

ANGDEG(ANG1(-THREE*TWOPI-HAFPI)) =  270.000000000000
ANGDEG(ANG2( THREE*TWOPI-HAFPI)) =  -90.000000000000

       ANGDEG(ATAN1(-SIX,SIX)) =  315.000000000000
       ANGDEG(ATAN2(-SIX,SIX)) =  -45.000000000000

          HMS( -36385.874L0 ) = -1006.258740
          HMS(  36385.874L0 ) =  1006.258740

        SECS( -1006.25874L0 ) = -36385.874000
        SECS(  1006.25874L0 ) =  36385.874000

    JULIAN_DAYNUM( 1980, 4, 2 ) =  2444332
```

```
  "Sam Wilson"
          -29
"Ben Wheeler"
          "N"
254.11112000
```

START_RANDOM_NUMBER_SEQUENCE (1)

Pseudorandom integers:

```
     16807
 282475249
1622650073
 984943658
1144108930
 470211272
 101027544
1457850878
1458777923
2007237709
```

START_RANDOM_NUMBER_SEQUENCE (2147483646)

Pseudorandom integers:

```
2147466840
1865008398
 524833574
1162539989
1003374717
1677272375
2046456103
 689632769
 688705724
 140245938
```

TEST_UTILVEMQ_MODULE

```
            V =    2.0000000000000   -6.0000000000000    3.0000000000000

            W =    4.0000000000000    5.0000000000000   -1.0000000000000

    DOTP(V,W) =  -25.0000000000000

      VMAG(V) =    7.0000000000000

  SXV(TWO,V) =    4.0000000000000  -12.0000000000000    6.0000000000000

    CRSP(V,W) =   -9.0000000000000   14.0000000000000   34.0000000000000

    VDIF(V,W) =   -2.0000000000000  -11.0000000000000    4.0000000000000

    VSUM(V,W) =    6.0000000000000   -1.0000000000000    2.0000000000000



            D =   30.0000000000000   10.0000000000000   40.0000000000000

     VXD(V,D) =   60.0000000000000  -60.0000000000000   120.000000000000



            M =    1.0000000000000    3.0000000000000    9.0000000000000
                   4.0000000000000    5.0000000000000    6.0000000000000
                   7.0000000000000    8.0000000000000    2.0000000000000

     VXM(V,M) =   -1.0000000000000    0.0000000000000  -12.0000000000000

    VXMT(V,M) =   11.0000000000000   -4.0000000000000  -28.0000000000000
```

```
                L =     1.0000000000000     2.0000000000000    -3.0000000000000
                       -2.0000000000000     5.0000000000000     6.0000000000000
                        4.0000000000000     3.0000000000000    -4.0000000000000

        MDIF(L,M) =     0.0000000000000    -1.0000000000000   -12.0000000000000
                       -6.0000000000000     0.0000000000000     0.0000000000000
                       -3.0000000000000    -5.0000000000000    -6.0000000000000

        MSUM(L,M) =     2.0000000000000     5.0000000000000     6.0000000000000
                        2.0000000000000    10.0000000000000    12.0000000000000
                       11.0000000000000    11.0000000000000    -2.0000000000000

         MXM(L,M) =   -12.0000000000000   -11.0000000000000    15.0000000000000
                       60.0000000000000    67.0000000000000    24.0000000000000
                      -12.0000000000000    -5.0000000000000    46.0000000000000

        MXMT(L,M) =   -20.0000000000000    -4.0000000000000    17.0000000000000
                       67.0000000000000    53.0000000000000    38.0000000000000
                      -23.0000000000000     7.0000000000000    44.0000000000000

        MTXM(L,M) =    21.0000000000000    25.0000000000000     5.0000000000000
                       43.0000000000000    55.0000000000000    54.0000000000000
                       -7.0000000000000   -11.0000000000000     1.0000000000000

          MINV(M) =    -1.0270270270270     1.7837837837838    -0.7297297297297
                        0.9189189189189    -1.6486486486486     0.8108108108108
                       -0.0810810810811     0.3513513513514    -0.1891891891892

N = MXM(M,MINV(M)) =    1.0000000000000     0.0000000000000     0.0000000000000
                        0.0000000000000     1.0000000000000     0.0000000000000
                        0.0000000000000     0.0000000000000     1.0000000000000

   MDIF(N,IDN3X3) =            -3.3E-016           0.00E+000           0.00E+000
                               1.11E-016           4.44E-016           0.00E+000
                               2.78E-017           -7.8E-016           4.44E-016
```

```
    X = CRSP(W,V) =     9.0000000000000  -14.0000000000000  -34.0000000000000

          N[3] = SXV( ONE/VMAG( X ), X )
          N[1] = SXV( ONE/VMAG( W ), W )
          N[2] = CRSP( N[3], N[1] )

             N =    0.6172133998484    0.7715167498105   -0.1543033499621
                    0.7500152304084   -0.5176735557710    0.4116931427785
                    0.2377493915935   -0.3698323869232   -0.8981643682420

             Q = IMATQ( N )

           Q.S =    0.2243743946150
           Q.V =    0.8707828839414    0.4368287457982    0.0239571893207

             P = QCXQ( Q, Q )

           P.S =    1.0000000000000
           P.V =    0.0000000000000    0.0000000000000    0.0000000000000

     ONE - P.S =          -2.2E-016
           P.V =          0.00E+000          0.00E+000          0.00E+000

  N = MXMT( N, N ) =    1.0000000000000    0.0000000000000    0.0000000000000
                    0.0000000000000    1.0000000000000    0.0000000000000
                    0.0000000000000    0.0000000000000    1.0000000000000

   MDIF(N,IDN3X3) =          0.00E+000          0.00E+000          -2.8E-017
                          0.00E+000          0.00E+000          0.00E+000
                          -2.8E-017          0.00E+000          -2.2E-016
```

```
                    PRY =  -145.000000000000   65.0000000000000  -170.000000000000

                    RPR =    10.0000000000000  -15.0000000000000   20.0000000000000

                      P = PRYQ( EULRAD( PRY ) )
                      Q = RPRQ( EULRAD( RPR ) )

                    P.S =     0.5325855897693
                    P.V =     0.8153775734730    0.0908499818748   -0.2079862568553

                    Q.S =     0.9576621969425
                    Q.V =     0.2566048122926   -0.1300295006517    0.0113761072310

EULDEG( QPRY( P ) ) =  -145.000000000000   65.0000000000000  -170.000000000000

EULDEG( QRPR( Q ) ) =  -170.000000000000   15.0000000000000  -160.000000000000



                    PYR =    80.0000000000000  -35.0000000000000   120.000000000000

                    YRY =    -5.0000000000000   80.0000000000000  -55.0000000000000

                      P = PYRQ( EULRAD( PYR ) )
                      Q = YRYQ( EULRAD( YRY ) )

                    P.S =     0.5326888026743
                    P.V =     0.5360641466904    0.1070262978581   -0.6460829990840

                    Q.S =     0.6634139481689
                    Q.V =     0.5825634160696    0.2716537822742   -0.3830222215595

EULDEG( QPYR( P ) ) =    80.0000000000000  -35.0000000000000   120.000000000000

EULDEG( QYRY( Q ) ) =    -5.0000000000000   80.0000000000000  -55.0000000000000
```

```
              R = QCXQ( P, Q )

            R.S =     0.9422227864928
            R.V =    -0.1798248634331    0.2447640278487    0.1413145773664

              R = QXQC( P, Q )

            R.S =     0.9422227864928
            R.V =    -0.0892100165536    0.0973556494454   -0.3078630719675

              R = QXQ( P, Q )

            R.S =    -0.2354364230378
            R.V =     0.8004748806090    0.0446498281944   -0.5493778745667

      QMAT( R ) =     0.3923806875575   -0.1872049915819   -0.9005507687847
                      0.3301692551544   -0.8851521670988    0.3278626298179
                     -0.8585019854273   -0.4259811406708   -0.2855072832874

              N = MDIF( QMAT(R), MXM(QMAT(P),QMAT(Q)) )

              N =           1.11E-016          1.67E-016          1.11E-016
                            0.00E+000          0.00E+000          0.00E+000
                            2.22E-016          1.67E-016          -1.1E-016

       ROT(V,Q) =    -0.1519683046476    3.1770527943276    6.2356428037884

      IROT(V,Q) =    -4.0877962276343   -3.4935338305642   -4.4816451640041

              X = VDIF( V, IROT(ROT(V,Q),Q) )

              X =           0.00E+000          -1.8E-015          8.88E-016
```

```
          P.S = PI * R.S
          P.V = SXV( PI, R.V )

          P.S =   -0.7396453370029
          P.V =    2.5147660043043    0.1402715722396   -1.7259214947836

            Q = UNIQUAT( P )

          Q.S =   -0.2354364230378
          Q.V =    0.8004748806090    0.0446498281944   -0.5493778745667

            R = QXQC( Q, Q )

    ONE - R.S =          0.00E+000
          R.V =          0.00E+000          0.00E+000          0.00E+000




        RPRPB =   10.0000000000000  -15.0000000000000   20.0000000000000

    NERTENS_P =   25.0000000000000    0.0000000000000    0.0000000000000
                   0.0000000000000   40.0000000000000    0.0000000000000
                   0.0000000000000    0.0000000000000   55.0000000000000

          MPB = QMAT(RPRQ(EULRAD( RPRPB )))
    NERTENS_B = MTXM( MPB,MXM( NERTENS_P, MPB))

    NERTENS_B =   26.9793202303121    3.1503479728235    6.7143670804884
                   3.1503479728235   43.4622804063605    5.7920928450464
                   6.7143670804884    5.7920928450464   49.5583993633275

            S =  NERTENS_B[1,1],
                 NERTENS_B[2,1], NERTENS_B[2,2],
                 NERTENS_B[3,1], NERTENS_B[3,2], NERTENS_B[3,3]

            DIAGONALIZE ( S, 1.0L-8, E, N )

            E =   25.0000000000000   40.0000000000000   55.0000000000000

        RPRBP = EULDEG( QRPR( IMATQ( N ) ) )

        RPRBP =  -20.0000000000000   15.0000000000000   -9.9999999957838
```

Test UNIFORM_RANDOM_SCALAR Function

CUMULATIVE DISTRIBUTION OF 1000 PSEUDORANDOM NUMBERS
FROM A
UNIFORMLY DISTRIBUTED POPULATION
HAVING ZERO MEAN
(UNCERTAINTY = 5.0)

CPU Time = 2.20 sec

CLASS UPPER BOUND	ACTUAL DISTRIBUTION	EXPECTED DISTRIBUTION	ACTUAL / EXPECTED
-5.0	0	0	1.0000
-4.0	103	100	1.0300
-3.0	215	200	1.0750
-2.0	328	300	1.0933
-1.0	412	400	1.0300
0.0	513	500	1.0260
1.0	602	600	1.0033
2.0	704	700	1.0057
3.0	809	800	1.0113
4.0	912	900	1.0133
5.0	1000	1000	1.0000

Test GAUSSIAN_RANDOM_SCALAR Function

CUMULATIVE DISTRIBUTION OF 1000 PSEUDORANDOM NUMBERS
FROM A
NORMALLY DISTRIBUTED POPULATION
HAVING ZERO MEAN
(SIGMA = 10.0)

CPU Time =   5.81 sec

CLASS UPPER BOUND	ACTUAL DISTRIBUTION	EXPECTED DISTRIBUTION	ACTUAL / EXPECTED
-25.0	6	6	1.0000
-20.0	21	23	0.9130
-15.0	63	67	0.9403
-10.0	146	159	0.9182
-5.0	298	309	0.9644
0.0	485	500	0.9700
5.0	687	691	0.9942
10.0	841	841	1.0000
15.0	937	933	1.0043
20.0	981	977	1.0041
25.0	994	994	1.0000

Test SIXTUC_MATRIX Function

```
SIXPOPIN  =     1.000
                0.487    1.000
                0.535    0.699    1.000
               -0.478   -0.996   -0.706    1.000
               -0.199   -0.662   -0.627    0.667    1.000
               -0.460   -0.778   -0.871    0.789    0.720    1.000


SIXPOPOUT = T * M ,  where M is SIXTUC_MATRIX( SIXPOPIN )
                     and T is the transpose of M.


SIXPOPOUT =     1.000
                0.487    1.000
                0.535    0.699    1.000
               -0.478   -0.996   -0.706    1.000
               -0.199   -0.662   -0.627    0.667    1.000
               -0.460   -0.778   -0.871    0.789    0.720    1.000


|SIXPOPOUT-SIXPOPIN| = 0.0E+000
                       0.0E+000 0.0E+000
                       0.0E+000 0.0E+000 0.0E+000
                       0.0E+000 0.0E+000 0.0E+000 0.0E+000
                       0.0E+000 0.0E+000 0.0E+000 0.0E+000 2.2E-016
                       0.0E+000 0.0E+000 1.1E-016 1.1E-016 1.1E-016 0.0E+000
```

Test GAUSSIAN_RANDOM_SIXVECTOR Function

ASTP / Apollo STDN 2-Station Estimation Error

```
SIXPOPIN  =   75.301
               0.487  179.906
               0.535    0.699   79.855
              -0.478   -0.996   -0.706    0.190
              -0.199   -0.662   -0.627    0.667    0.114
              -0.460   -0.778   -0.871    0.789    0.720    0.417
```

SIXPOPOUT = statistical summary of 1000 pseudorandom
six-vectors from population defined by SIXPOPIN

CPU time = 64.13 sec

```
SIXPOPOUT =   76.626
               0.504  183.598
               0.567    0.711   79.939
              -0.498   -0.996   -0.720    0.193
              -0.222   -0.671   -0.640    0.677    0.113
              -0.473   -0.787   -0.875    0.798    0.729    0.415
```

Test GAUSSIAN_RANDOM_SIXVECTOR Function

PAIDS / Long-Range Stationkeep without TIC ; 0.5 deg Deadbands

SIXPOPIN =

10.100					
0.376	40.300				
-0.192	0.165	11.900			
-0.405	-0.652	-0.088	0.063		
0.553	-0.052	-0.358	0.068	0.073	
-0.341	0.274	0.187	-0.208	-0.361	0.026

SIXPOPOUT = statistical summary of 1000 pseudorandom six-vectors from population defined by SIXPOPIN

CPU time = 64.10 sec

SIXPOPOUT =

10.162					
0.345	38.759				
-0.201	0.126	11.897			
-0.398	-0.607	-0.025	0.061		
0.576	-0.019	-0.345	0.052	0.072	
-0.414	0.237	0.198	-0.141	-0.389	0.026

Test GAUSSIAN_RANDOM_SIXVECTOR Function

PAIDS / Long-Range Stationkeep with TIC ; 3.0 deg Deadbands

SIXPOPIN =

6.900					
0.431	27.800				
0.301	0.098	6.900			
-0.467	-0.360	0.332	0.042		
0.519	0.515	0.329	0.104	0.063	
-0.072	0.056	0.363	0.068	-0.015	0.024

SIXPOPOUT = statistical summary of 1000 pseudorandom
six-vectors from population defined by SIXPOPIN

CPU time = 64.10 sec

SIXPOPOUT =

6.884					
0.422	27.852				
0.331	0.101	6.925			
-0.489	-0.311	0.286	0.042		
0.523	0.544	0.336	0.107	0.063	
-0.042	0.067	0.411	0.102	0.002	0.024

Tests completed

Elapsed time = 302.88 seconds

TEST_UTILMATH_MODULE

```
       INT( -2.3 ) = -3
       INT( -2.0 ) = -2
       INT( -0.3 ) = -1
       INT(  0.0 ) =  0
       INT(  1.3 ) =  1

      FRAC( -2.3 ) =  0.70
      FRAC( -2.0 ) =  0.00
      FRAC( -0.3 ) =  0.70
      FRAC(  0.0 ) =  0.00
      FRAC(  1.3 ) =  0.30

RMOD( -2.8, -0.5 ) = -0.30
RMOD( -2.8,  0.0 ) =  0.00
RMOD( -2.8,  0.5 ) =  0.20

RMOD(  2.8, -0.5 ) = -0.20
RMOD(  2.8,  0.0 ) =  0.00
RMOD(  2.8,  0.5 ) =  0.30

     RSIGN( -1.9 ) = -1
     RSIGN(  0.0 ) =  1
     RSIGN(  1.9 ) =  1

       ISIGN( -5 ) = -1
       ISIGN(  0 ) =  1
       ISIGN(  5 ) =  1

    IMAX( -3, -4 ) = -3
    IMAX(  3,  4 ) =  4

    IMIN( -3, -4 ) = -4
    IMIN(  3,  4 ) =  3

RMAX( -2.9, -3.9 ) = -2.90
RMAX(  2.9,  3.9 ) =  3.90

RMIN( -2.9, -3.9 ) = -3.90
RMIN(  2.9,  3.9 ) =  2.90
```

```
              ANGDEG(ONE) =  57.29577951308232
     ANGRAD(ANGDEG(ONE))-ONE =  0.0L+00

ANGDEG(ANG1(-THREE*TWOPI-HAFPI)) =  270.000000000000
ANGDEG(ANG2( THREE*TWOPI-HAFPI)) =  -90.000000000000

        ANGDEG(ATAN1(-SIX,SIX)) =  315.000000000000
        ANGDEG(ATAN2(-SIX,SIX)) =  -45.000000000000

          HMS( -36385.874L0 ) = -1006.258740
          HMS(  36385.874L0 ) =  1006.258740

         SECS( -1006.25874L0 ) = -36385.874000
         SECS(  1006.25874L0 ) =  36385.874000

     JULIAN_DAYNUM( 1980, 4, 2 ) =  2444332
```

TEST_UTILSPIF_MODULE

```
  "Sam Wilson"
         -29
 "Ben Wheeler"
          "N"
254.11112000
```

START_RANDOM_NUMBER_SEQUENCE (1)

Pseudorandom integers:

```
     16807
 282475249
1622650073
 984943658
1144108930
 470211272
 101027544
1457850878
1458777923
2007237709
```

START_RANDOM_NUMBER_SEQUENCE (2147483646)

Pseudorandom integers:

```
2147466840
1865008398
 524833574
1162539989
1003374717
1677272375
2046456103
 689632769
 688705724
 140245938
```

TEST_UTILVEMQ_MODULE

```
          V =     2.0000000000000    -6.0000000000000     3.0000000000000

          W =     4.0000000000000     5.0000000000000    -1.0000000000000

  DOTP(V,W) =   -25.0000000000000

    VMAG(V) =     7.0000000000000

 SXV(TWO,V) =     4.0000000000000   -12.0000000000000     6.0000000000000

  CRSP(V,W) =    -9.0000000000000    14.0000000000000    34.0000000000000

  VDIF(V,W) =    -2.0000000000000   -11.0000000000000     4.0000000000000

  VSUM(V,W) =     6.0000000000000    -1.0000000000000     2.0000000000000



          D =    30.0000000000000    10.0000000000000    40.0000000000000

   VXD(V,D) =    60.0000000000000   -60.0000000000000   120.0000000000000



          M =     1.0000000000000     3.0000000000000     9.0000000000000
                  4.0000000000000     5.0000000000000     6.0000000000000
                  7.0000000000000     8.0000000000000     2.0000000000000

   VXM(V,M) =    -1.0000000000000     0.0000000000000   -12.0000000000000

  VXMT(V,M) =    11.0000000000000    -4.0000000000000   -28.0000000000000
```

```
                   L =   1.0000000000000    2.0000000000000   -3.0000000000000
                        -2.0000000000000    5.0000000000000    6.0000000000000
                         4.0000000000000    3.0000000000000   -4.0000000000000

           MDIF(L,M) =   0.0000000000000   -1.0000000000000  -12.0000000000000
                        -6.0000000000000    0.0000000000000    0.0000000000000
                        -3.0000000000000   -5.0000000000000   -6.0000000000000

           MSUM(L,M) =   2.0000000000000    5.0000000000000    6.0000000000000
                         2.0000000000000   10.0000000000000   12.0000000000000
                        11.0000000000000   11.0000000000000   -2.0000000000000

            MXM(L,M) = -12.0000000000000  -11.0000000000000   15.0000000000000
                        60.0000000000000   67.0000000000000   24.0000000000000
                       -12.0000000000000   -5.0000000000000   46.0000000000000

           MXMT(L,M) = -20.0000000000000   -4.0000000000000   17.0000000000000
                        67.0000000000000   53.0000000000000   38.0000000000000
                       -23.0000000000000    7.0000000000000   44.0000000000000

           MTXM(L,M) =  21.0000000000000   25.0000000000000    5.0000000000000
                        43.0000000000000   55.0000000000000   54.0000000000000
                        -7.0000000000000  -11.0000000000000    1.0000000000000

             MINV(M) =  -1.0270270270270    1.7837837837838   -0.7297297297297
                         0.9189189189189   -1.6486486486486    0.8108108108108
                        -0.0810810810811    0.3513513513514   -0.1891891891892

N = MXM(M,MINV(M)) =     1.0000000000000    0.0000000000000    0.0000000000000
                         0.0000000000000    1.0000000000000    0.0000000000000
                       2.77555756156L-17   -0.0000000000000    1.0000000000000

     MDIF(N,IDN3X3) =            -3.3L-16            0.0L+00            0.0L+00
                                  1.1L-16            4.4L-16            0.0L+00
                                  2.8L-17           -7.8L-16            4.4L-16
```

```
       X = CRSP(W,V) =     9.0000000000000  -14.0000000000000  -34.0000000000000

               N[3] = SXV( ONE/VMAG( X ), X )
               N[1] = SXV( ONE/VMAG( W ), W )
               N[2] = CRSP( N[3], N[1] )

                  N =     0.6172133998484    0.7715167498105   -0.1543033499621
                          0.7500152304084   -0.5176735557710    0.4116931427785
                          0.2377493915935   -0.3698323869232   -0.8981643682420

                  Q = IMATQ( N )

                Q.S =     0.2243743946150
                Q.V =     0.8707828839414    0.4368287457982    0.0239571893207

                  P = QCXQ( Q, Q )

                P.S =     1.0000000000000
                P.V =     0.0000000000000    0.0000000000000    0.0000000000000

         ONE - P.S =             -2.2L-16
                P.V =              0.0L+00            0.0L+00            0.0L+00

    N = MXMT( N, N ) =     1.0000000000000  5.55111512313L-17 -5.55111512313L-17
                        5.55111512313L-17    1.0000000000000  5.55111512313L-17
                       -5.55111512313L-17  5.55111512313L-17    1.0000000000000

      MDIF(N,IDN3X3) =              0.0L+00            5.6L-17           -5.6L-17
                                    5.6L-17            0.0L+00            5.6L-17
                                   -5.6L-17            5.6L-17            0.0L+00
```

```
                    PRY = -145.0000000000000   65.0000000000000 -170.0000000000000

                    RPR =   10.0000000000000  -15.0000000000000   20.0000000000000

                      P = PRYQ( EULRAD( PRY ) )
                      Q = RPRQ( EULRAD( RPR ) )

                    P.S =    0.5325855897693
                    P.V =    0.8153775734730    0.0908499818748   -0.2079862568553

                    Q.S =    0.9576621969425
                    Q.V =    0.2566048122926   -0.1300295006517    0.0113761072310

EULDEG( QPRY( P ) ) = -145.0000000000000   65.0000000000000 -170.0000000000000

EULDEG( QRPR( Q ) ) = -170.0000000000000   15.0000000000000 -160.0000000000000



                    PYR =   80.0000000000000  -35.0000000000000  120.0000000000000

                    YRY =   -5.0000000000000   80.0000000000000  -55.0000000000000

                      P = PYRQ( EULRAD( PYR ) )
                      Q = YRYQ( EULRAD( YRY ) )

                    P.S =    0.5326888026743
                    P.V =    0.5360641466904    0.1070262978581   -0.6460829990840

                    Q.S =    0.6634139481689
                    Q.V =    0.5825634160696    0.2716537822742   -0.3830222215595

EULDEG( QPYR( P ) ) =   80.0000000000000  -35.0000000000000  120.0000000000000

EULDEG( QYRY( Q ) ) =   -5.0000000000000   80.0000000000000  -55.0000000000000
```

```
     R = QCXQ( P, Q )

   R.S =    0.9422227864928
   R.V =   -0.1798248634331     0.2447640278487     0.1413145773664

     R = QXQC( P, Q )

   R.S =    0.9422227864928
   R.V =   -0.0892100165536     0.0973556494454    -0.3078630719675

     R = QXQ( P, Q )

   R.S =   -0.2354364230378
   R.V =    0.8004748806090     0.0446498281944    -0.5493778745667

QMAT( R ) =    0.3923806875575    -0.1872049915819    -0.9005507687847
               0.3301692551544    -0.8851521670988     0.3278626298179
              -0.8585019854273    -0.4259811406708    -0.2855072832874

     N = MDIF( QMAT(R), MXM(QMAT(P),QMAT(Q)) )

     N =              5.6L-17            -5.6L-17            -1.1L-16
                      0.0L+00            -2.2L-16             5.6L-17
                     -1.1L-16            -5.6L-17            -2.8L-16

 ROT(V,Q) =   -0.1519683046476     3.1770527943276     6.2356428037884

IROT(V,Q) =   -4.0877962276343    -3.4935338305642    -4.4816451640041

     X = VDIF( V, IROT(ROT(V,Q),Q) )

     X =              4.4L-16             0.0L+00             0.0L+00
```

```
           P.S = PI * R.S
           P.V = SXV( PI, R.V )

           P.S =   -0.7396453370029
           P.V =    2.5147660043043    0.1402715722396   -1.7259214947836

             Q = UNIQUAT( P )

           Q.S =   -0.2354364230378
           Q.V =    0.8004748806090    0.0446498281944   -0.5493778745667

             R = QXQC( Q, Q )

   ONE - R.S =            2.2L-16
           R.V =            0.0L+00            0.0L+00            0.0L+00



         RPRPB =   10.0000000000000  -15.0000000000000   20.0000000000000

    NERTENS_P =   25.0000000000000    0.0000000000000    0.0000000000000
                   0.0000000000000   40.0000000000000    0.0000000000000
                   0.0000000000000    0.0000000000000   55.0000000000000

           MPB = QMAT(RPRQ(EULRAD( RPRPB )))
    NERTENS_B = MTXM( MPB,MXM( NERTENS_P, MPB))

    NERTENS_B =   26.9793202303121    3.1503479728235    6.7143670804884
                   3.1503479728235   43.4622804063605    5.7920928450464
                   6.7143670804884    5.7920928450464   49.5583993633275

             S =  NERTENS_B[1,1],
                  NERTENS_B[2,1], NERTENS_B[2,2],
                  NERTENS_B[3,1], NERTENS_B[3,2], NERTENS_B[3,3]

             DIAGONALIZE ( S, 1.0L-8, E, N )

             E =   25.0000000000000   40.0000000000000   55.0000000000000

         RPRBP = EULDEG( QRPR( IMATQ( N ) ) )

         RPRBP =  -20.0000000000000   15.0000000000000   -9.9999999957838
```

Test UNIFORM_RANDOM_SCALAR Function

CUMULATIVE DISTRIBUTION OF 1000 PSEUDORANDOM NUMBERS
FROM A
UNIFORMLY DISTRIBUTED POPULATION
HAVING ZERO MEAN
(UNCERTAINTY = 5.0)

CPU Time = 0.80 sec

CLASS UPPER BOUND	ACTUAL DISTRIBUTION	EXPECTED DISTRIBUTION	ACTUAL / EXPECTED
-5.0	0	0	1.0000
-4.0	103	100	1.0300
-3.0	215	200	1.0750
-2.0	328	300	1.0933
-1.0	412	400	1.0300
0.0	513	500	1.0260
1.0	602	600	1.0033
2.0	704	700	1.0057
3.0	809	800	1.0113
4.0	912	900	1.0133
5.0	1000	1000	1.0000

Test GAUSSIAN_RANDOM_SCALAR Function

CUMULATIVE DISTRIBUTION OF 1000 PSEUDORANDOM NUMBERS
FROM A
NORMALLY DISTRIBUTED POPULATION
HAVING ZERO MEAN
(SIGMA = 10.0)

CPU Time = 2.83 sec

CLASS UPPER BOUND	ACTUAL DISTRIBUTION	EXPECTED DISTRIBUTION	ACTUAL / EXPECTED
-25.0	6	6	1.0000
-20.0	21	23	0.9130
-15.0	63	67	0.9403
-10.0	146	159	0.9182
-5.0	298	309	0.9644
0.0	485	500	0.9700
5.0	687	691	0.9942
10.0	841	841	1.0000
15.0	937	933	1.0043
20.0	981	977	1.0041
25.0	994	994	1.0000

Test SIXTUC_MATRIX Function

```
        SIXPOPIN  =     1.000
                        0.487     1.000
                        0.535     0.699     1.000
                       -0.478    -0.996    -0.706     1.000
                       -0.199    -0.662    -0.627     0.667     1.000
                       -0.460    -0.778    -0.871     0.789     0.720     1.000


        SIXPOPOUT = T * M ,  where M is SIXTUC_MATRIX( SIXPOPIN )
                             and T is the transpose of M.


        SIXPOPOUT =     1.000
                        0.487     1.000
                        0.535     0.699     1.000
                       -0.478    -0.996    -0.706     1.000
                       -0.199    -0.662    -0.627     0.667     1.000
                       -0.460    -0.778    -0.871     0.789     0.720     1.000


|SIXPOPOUT-SIXPOPIN| =  2.2L-16
                        0.0L+00  0.0L+00
                        0.0L+00  0.0L+00  1.1L-16
                        0.0L+00  0.0L+00  0.0L+00  0.0L+00
                        0.0L+00  0.0L+00  0.0L+00  0.0L+00  0.0L+00
                        0.0L+00  0.0L+00  0.0L+00  0.0L+00  1.1L-16  0.0L+00
```

```
                Test GAUSSIAN_RANDOM_SIXVECTOR Function


        ASTP  / Apollo STDN 2-Station Estimation Error


        SIXPOPIN  =   75.301
                       0.487  179.906
                       0.535    0.699   79.855
                      -0.478   -0.996   -0.706    0.190
                      -0.199   -0.662   -0.627    0.667    0.114
                      -0.460   -0.778   -0.871    0.789    0.720    0.417


        SIXPOPOUT = statistical summary of 1000 pseudorandom
                    six-vectors from population defined by SIXPOPIN


         CPU time = 18.42 sec


        SIXPOPOUT =   76.626
                       0.504  183.598
                       0.567    0.711   79.939
                      -0.498   -0.996   -0.720    0.193
                      -0.222   -0.671   -0.640    0.677    0.113
                      -0.473   -0.787   -0.875    0.798    0.729    0.415
```

Test GAUSSIAN_RANDOM_SIXVECTOR Function

PAIDS / Long-Range Stationkeep without TIC ; 0.5 deg Deadbands

```
SIXPOPIN  =   10.100
               0.376   40.300
              -0.192    0.165   11.900
              -0.405   -0.652   -0.088    0.063
               0.553   -0.052   -0.358    0.068    0.073
              -0.341    0.274    0.187   -0.208   -0.361    0.026
```

SIXPOPOUT = statistical summary of 1000 pseudorandom six-vectors from population defined by SIXPOPIN

CPU time = 18.40 sec

```
SIXPOPOUT =   10.162
               0.345   38.759
              -0.201    0.126   11.897
              -0.398   -0.607   -0.025    0.061
               0.576   -0.019   -0.345    0.052    0.072
              -0.414    0.237    0.198   -0.141   -0.389    0.026
```

Test GAUSSIAN_RANDOM_SIXVECTOR Function

PAIDS / Long-Range Stationkeep with TIC ; 3.0 deg Deadbands

```
SIXPOPIN  =    6.900
               0.431   27.800
               0.301    0.098    6.900
              -0.467   -0.360    0.332    0.042
               0.519    0.515    0.329    0.104    0.063
              -0.072    0.056    0.363    0.068   -0.015    0.024
```

SIXPOPOUT = statistical summary of 1000 pseudorandom
six-vectors from population defined by SIXPOPIN

CPU time = 18.38 sec

```
SIXPOPOUT =    6.884
               0.422   27.852
               0.331    0.101    6.925
              -0.489   -0.311    0.286    0.042
               0.523    0.544    0.336    0.107    0.063
              -0.042    0.067    0.411    0.102    0.002    0.024
```

Tests completed

Elapsed time = 150.58 seconds