Basic Interval Arithmetic Module

The following Fortran 90 module implements some basic items to show proof of concept for a specific proposal for interval arithmetic.

```
MODULE WG5BASIC

!**********************************************************************
!**                                                                  **
!**  This module implements some basic items to show proof of        **
!**  concept for a specific proposal for interval arithmetic in      **
!**  Fortran 90.  The ideas resulted from group discussions by       **
!**  Keith Bierman, George Corliss, David Hough, Baker Kearfott,     **
!**  Andrew Pitonyak, Michael Schulte, William Walster, and          **
!**  Wolfgang Walter.                                                **
!**                                                                  **
!**********************************************************************
!**                                                                  **
!**    Copyright (c) 1996 by Andrew Pitonyak and Wolfgang Walter     **
!**                                                                  **
!**    This code was created as a demonstration package for the      **
!**    ISO/IEC JTC1/SC22/WG5 committee and may be freely used        **
!**    and modified by the members of this committee.                **
!**                                                                  **
!**********************************************************************
!**                                                                  **
!**                           Notes:                                 **
!**                                                                  **
!**    It is assumed that all Fortran intrinsic operations and       **
!**    functions with floating point operands or arguments return    **
!**    results accurate to within one unit in the last place (ulp)   **
!**    accuracy.                                                     **
!**                                                                  **
!**********************************************************************
!**                                                                  **
!**  The primary author of this code is Andrew D. Pitonyak           **
!**  Primary:   pitonyak@math.tu-dresden.de                          **
!**  Secondary: apitonya@oakland.edu                                 **
!**                                                                  **
!**             Andrew Pitonyak                                      **
!**             Inst. f. Wissenschaftliches Rechnen                  **
!**             TU Dresden                                           **
!**        DE - 01062 Dresden                                        **
!**             Germany                                              **
!**                                                                  **
!**********************************************************************
!**                                                                  **
!**                          Module History                          **
!**                                                                  **
!**  June 30, 1996: Started coding.                                  **
```

```
!**  July 19, 1996: Stopped coding. (I am not the Energizer Bunny)  **
!**                                                                  **
!**********************************************************************

IMPLICIT NONE

    PRIVATE

    PUBLIC::    fpkind, INTERVAL, INF, SUP, MID, WID, MAG, MIG,  &
&               MIN, MAX, ABS, IVAL, SQRT,                       &
&               OPERATOR(+), OPERATOR(-), OPERATOR(*),           &
&               OPERATOR(/), OPERATOR(==), OPERATOR(/=),         &
&               OPERATOR(<), OPERATOR(>), OPERATOR(<=),          &
&               OPERATOR(>=), OPERATOR(.PLT.), OPERATOR(.PGT.),  &
&               OPERATOR(.PLE.), OPERATOR(.PGE.), OPERATOR(.IS.),&
&               OPERATOR(.CH.), OPERATOR(.SB.), OPERATOR(.SP.),  &
&               OPERATOR(.PSB.), OPERATOR(.PSP.), OPERATOR(.DJ.),&
&               OPERATOR(.IN.)

!**********************************************************************
!**                                                                  **
!**  First for the constants and the data type INTERVAL.             **
!**  By default, the system is setup to use double precision REALs   **
!**  for the components of the INTERVAL type. fpkind is a constant   **
!**  with the corresponding kind type parameter value. By changing   **
!**  its value and recompiling, the whole module will change         **
!**  precision.                                                      **
!**                                                                  **
!**********************************************************************

    INTEGER, PARAMETER      :: fpkind = KIND(0.0D0)

    TYPE :: INTERVAL
        PRIVATE
        REAL(fpkind) :: INF, SUP
    END TYPE INTERVAL

    REAL(fpkind), PARAMETER :: zero     = 0.0_fpkind,            &
                               one      = 1.0_fpkind,            &
                               two      = 2.0_fpkind,            &
                               up       = 1.0_fpkind,            &
                               down     =-1.0_fpkind

!**********************************************************************
!**                                                                  **
!**  The interface functions follow                                  **
!**                                                                  **
!**********************************************************************

    INTERFACE IVAL
        MODULE PROCEDURE IVAL
    END INTERFACE
```

```
    INTERFACE INF
        MODULE PROCEDURE INF
    END INTERFACE

    INTERFACE SUP
        MODULE PROCEDURE SUP
    END INTERFACE

    INTERFACE MID
        MODULE PROCEDURE MID
    END INTERFACE

    INTERFACE WID
        MODULE PROCEDURE WID
    END INTERFACE

    INTERFACE MAG
        MODULE PROCEDURE MAG
    END INTERFACE

    INTERFACE MIG
        MODULE PROCEDURE MIG
    END INTERFACE

    INTERFACE ABS
        MODULE PROCEDURE INTERVAL_ABS
    END INTERFACE

    INTERFACE MAX
        MODULE PROCEDURE INTERVAL_MAX
    END INTERFACE

    INTERFACE MIN
        MODULE PROCEDURE INTERVAL_MIN
    END INTERFACE

    INTERFACE SQRT
        MODULE PROCEDURE INTERVAL_SQRT
    END INTERFACE

!**********************************************************************
!**                                                                  **
!**  Now for some operators                                          **
!**                                                                  **
!**********************************************************************

    INTERFACE OPERATOR  ( + )
        MODULE PROCEDURE POS_I, I_ADD_I
    END INTERFACE

    INTERFACE OPERATOR  ( - )
```

```
    MODULE PROCEDURE NEG_I, I_SUB_I
END INTERFACE

INTERFACE OPERATOR  ( * )
    MODULE PROCEDURE I_MUL_I
END INTERFACE

INTERFACE OPERATOR  ( / )
    MODULE PROCEDURE I_DIV_I
END INTERFACE

INTERFACE OPERATOR  ( == )
    MODULE PROCEDURE I_EQ_I
END INTERFACE

INTERFACE OPERATOR  ( /= )
    MODULE PROCEDURE I_NE_I
END INTERFACE

INTERFACE OPERATOR  ( < )
    MODULE PROCEDURE I_LT_I
END INTERFACE

INTERFACE OPERATOR  ( > )
    MODULE PROCEDURE I_GT_I
END INTERFACE

INTERFACE OPERATOR  ( <= )
    MODULE PROCEDURE I_LE_I
END INTERFACE

INTERFACE OPERATOR  ( >= )
    MODULE PROCEDURE I_GE_I
END INTERFACE

INTERFACE OPERATOR  ( .PLT. )
    MODULE PROCEDURE I_PLT_I
END INTERFACE

INTERFACE OPERATOR  ( .PGT. )
    MODULE PROCEDURE I_PGT_I
END INTERFACE

INTERFACE OPERATOR  ( .PLE. )
    MODULE PROCEDURE I_PLE_I
END INTERFACE

INTERFACE OPERATOR  ( .PGE. )
    MODULE PROCEDURE I_PGE_I
END INTERFACE

INTERFACE OPERATOR  ( .IS. )
```

```
        MODULE PROCEDURE I_INTERSECT_I
    END INTERFACE

    INTERFACE OPERATOR  ( .CH. )
        MODULE PROCEDURE I_CONVEX_HULL_I
    END INTERFACE

    INTERFACE OPERATOR  ( .SB. )
        MODULE PROCEDURE I_SUBSET_I
    END INTERFACE

    INTERFACE OPERATOR  ( .PSB. )
        MODULE PROCEDURE I_PROPER_SUBSET_I
    END INTERFACE

    INTERFACE OPERATOR  ( .SP. )
        MODULE PROCEDURE I_SUPERSET_I
    END INTERFACE

    INTERFACE OPERATOR  ( .PSP. )
        MODULE PROCEDURE I_PROPER_SUPERSET_I
    END INTERFACE

    INTERFACE OPERATOR  ( .DJ. )
        MODULE PROCEDURE I_DISJOINT_I
    END INTERFACE

    INTERFACE OPERATOR  ( .IN. )
        MODULE PROCEDURE R_IN_I, I_IN_I
    END INTERFACE

!*****************************************************************
!**                                                             **
!**  Some special IEEE values are defined for internal use. Done  **
!**  properly these should simply be constants, but a bug in some  **
!**  compilers, most noteably for me, the one I use, has prevented  **
!**  this so far, so they are currently functions.              **
!**                                                             **
!*****************************************************************

! REAL(fpkind), SAVE :: IEEE_PLUSINFTY
! INTEGER, DIMENSION(2) :: PLUSINFTY
! EQUIVALENCE (IEEE_PLUSINFTY,PLUSINFTY )
! DATA PLUSINFTY / Z'7FF00000',Z'00000000'/
!
! REAL(fpkind), SAVE :: IEEE_MINUSINFTY
! INTEGER, DIMENSION(2) :: MINUSINFTY
! EQUIVALENCE (IEEE_MINUSINFTY,MINUSINFTY)
! DATA MINUSINFTY / Z'FFF00000',Z'00000000'/
!
! REAL(fpkind), SAVE :: IEEE_NAN
! INTEGER, DIMENSION(2) :: NAN
```

```
! EQUIVALENCE (IEEE_NAN,NAN)
! DATA NAN / Z'7FFFFFF',Z'FFFFFFFF'/ ! One possible NAN

    INTERFACE GET_IEEE_NAN
        MODULE PROCEDURE GET_IEEE_NAN
    END INTERFACE

    INTERFACE GET_IEEE_PLUSINFTY
        MODULE PROCEDURE GET_IEEE_PLUSINFTY
    END INTERFACE

    INTERFACE GET_IEEE_MINUSINFTY
        MODULE PROCEDURE GET_IEEE_MINUSINFTY
    END INTERFACE

    INTERFACE SIGNAL_ERROR
        MODULE PROCEDURE SIGNAL_ERROR
    END INTERFACE

CONTAINS


!*********************************************************************
!**                                                                 **
!**   FUNCTION: INF(X)                                              **
!**                                                                 **
!**   Input   : INTERVAL                                            **
!**                                                                 **
!**   Output  : REAL, The lower bound of the input interval         **
!**                                                                 **
!*********************************************************************

    REAL(fpkind) FUNCTION INF(X)
        TYPE(INTERVAL), INTENT(IN) :: X
        INF = X%INF
    END FUNCTION INF

!*********************************************************************
!**                                                                 **
!**   FUNCTION: SUP(X)                                              **
!**                                                                 **
!**   Input   : INTERVAL                                            **
!**                                                                 **
!**   Output  : REAL, The upper bound of the input interval         **
!**                                                                 **
!*********************************************************************

    REAL(fpkind) FUNCTION SUP(X)
        TYPE(INTERVAL), INTENT(IN) :: X
        SUP = X%SUP
    END FUNCTION SUP
```

```
!*********************************************************************
!**                                                                 **
!**   FUNCTION: MID(X)                                              **
!**                                                                 **
!**   Input   : INTERVAL                                            **
!**                                                                 **
!**   Output  : REAL, floating point approximation of the midpoint. **
!**             The interval will contain the returned midpoint.    **
!**                                                                 **
!**   Notes   : Overflow can happen.                                **
!**                                                                 **
!*********************************************************************

    REAL(fpkind) FUNCTION MID(X)
        TYPE(INTERVAL), INTENT(IN) :: X
        MID = X%INF + (X%SUP - X%INF) / TWO
        IF (MID < X%INF) THEN
            MID = X%INF
        ELSE IF (MID > X%SUP) THEN
            MID = X%SUP
        END IF
    END FUNCTION MID

!*********************************************************************
!**                                                                 **
!**   FUNCTION: WID(X)                                              **
!**                                                                 **
!**   Input   : INTERVAL                                            **
!**                                                                 **
!**   Output  : REAL, A floating point approximation to the width   **
!**             Although the result is rounded up, zero is returned **
!**             for point intervals.                                **
!**                                                                 **
!**   Notes   : This could be improved e.g.: if the signs and the   **
!**             exponents of the INF and the SUP are the same,      **
!**             simply subracting them should suffice.              **
!**                                                                 **
!*********************************************************************

    REAL(fpkind) FUNCTION WID(X)
        TYPE(INTERVAL), INTENT(IN) :: X
        IF (X%INF == X%SUP) THEN
            WID = zero
        ELSE
            WID = NEAREST(X%SUP - X%INF, UP)
        END IF
    END FUNCTION WID

!*********************************************************************
!**                                                                 **
!**   FUNCTION: MAG(X)                                              **
!**                                                                 **
```

```
!**   Input   : INTERVAL                                                **
!**                                                                     **
!**   Output  : REAL, maximum distance a value in the interval can      **
!**             be from zero. Think magnitude.                          **
!**                                                                     **
!*********************************************************************

    REAL(fpkind) FUNCTION MAG(X)
        TYPE(INTERVAL), INTENT(IN) :: X
        MAG = MAX(ABS(X%SUP),ABS(X%INF))
    END FUNCTION MAG

!*********************************************************************
!**                                                                   **
!**   Function: MIG(X)                                                **
!**                                                                   **
!**   Input   : INTERVAL                                              **
!**                                                                   **
!**   Output  : REAL, Minimum distance a value in the interval can    **
!**             be from zero. Think mignitude.                        **
!**                                                                   **
!*********************************************************************

    REAL(fpkind) FUNCTION MIG(X)
        TYPE(INTERVAL), INTENT(IN) :: X
        IF (zero .IN. X) THEN
            MIG = zero
        ELSE
            MIG = MIN(ABS(X%SUP),ABS(X%INF))
        END IF
    END FUNCTION MIG

!*********************************************************************
!**                                                                   **
!**   FUNCTION: INTERVAL_ABS(X)                                       **
!**                                                                   **
!**   Input   : INTERVAL                                              **
!**                                                                   **
!**   Output  : INTERVAL, The range of absolute values.               **
!**                                                                   **
!**   Notes   : The name INTERVAL_ABS is used to make it clear that   **
!**             an INTERVAL and not the largest absolute value is     **
!**             returned. The result interval will contain the        **
!**             absolute value of any point in the argument           **
!**             interval.                                             **
!**                                                                   **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION INTERVAL_ABS(X)
        TYPE(INTERVAL), INTENT(IN) :: X
        INTERVAL_ABS%INF = MIG(X)
        INTERVAL_ABS%SUP = MAG(X)
```

```
    END FUNCTION INTERVAL_ABS

!*********************************************************************
!**                                                                 **
!**   FUNCTION: INTERVAL_MAX(A1,A2)                                 **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : INTERVAL, The range of maximum values when          **
!**             arbitrarily picking pairs of values from the two    **
!**             argument intervals.                                 **
!**             Practically speaking, given two intervals <xl,xu>   **
!**             and <yl,yu> the resulting interval is given by      **
!**             <max(xl,yl), max(xu,yu)>.                           **
!**                                                                 **
!**   Notes   : This needs to be extended for multiple parameters   **
!**                                                                 **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION INTERVAL_MAX(A1, A2)
        TYPE(INTERVAL), INTENT(IN) :: A1, A2
        INTERVAL_MAX%INF = MAX(A1%INF, A2%INF)
        INTERVAL_MAX%SUP = MAX(A1%SUP, A2%SUP)
    END FUNCTION INTERVAL_MAX

!*********************************************************************
!**                                                                 **
!**   FUNCTION: INTERVAL_MIN(A1,A2)                                 **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : INTERVAL, The range of minimum values when          **
!**             arbitrarily picking pairs of values from the two    **
!**             argument intervals.                                 **
!**             Practically speaking, given two intervals <xl,xu>   **
!**             and <yl,yu> the resulting interval is given by      **
!**             <min(xl,yl), min(xu,yu)>.                           **
!**                                                                 **
!**   Notes   : This needs to be extended for multiple parameters   **
!**                                                                 **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION INTERVAL_MIN(A1, A2)
        TYPE(INTERVAL), INTENT(IN) :: A1, A2
        INTERVAL_MIN%INF = MIN(A1%INF, A2%INF)
        INTERVAL_MIN%SUP = MIN(A1%SUP, A2%SUP)
    END FUNCTION INTERVAL_MIN

!*********************************************************************
!**                                                                 **
!**   FUNCTION: INTERVAL_SQRT(X)                                    **
!**                                                                 **
```

```
!**   Input   : INTERVAL                                                **
!**                                                                     **
!**   Output  : INTERVAL, SQRT of the input interval.                   **
!**                                                                     **
!*********************************************************************

TYPE(INTERVAL) FUNCTION INTERVAL_SQRT(X)
    TYPE(INTERVAL), INTENT(IN) :: X
    IF (X%INF < zero) THEN
        CALL SIGNAL_ERROR("Argument to SQRT contains negative elements")
    ELSE
        INTERVAL_SQRT%INF = NEAREST(SQRT(X%INF), DOWN)
        INTERVAL_SQRT%SUP = NEAREST(SQRT(X%SUP), UP)
        IF (INTERVAL_SQRT%INF < zero) INTERVAL_SQRT%INF = zero
    END IF
 END FUNCTION INTERVAL_SQRT


!*********************************************************************
!**                                                                   **
!**   FUNCTION: GET_IEEE_PLUSINFTY()                                  **
!**                                                                   **
!**   Input   : None.                                                 **
!**                                                                   **
!**   Output  : REAL, An IEEE representation of positive infinity.    **
!**                                                                   **
!*********************************************************************

    REAL(fpkind) FUNCTION GET_IEEE_PLUSINFTY()
        REAL(fpkind), SAVE :: IEEE_PLUSINFTY
        INTEGER, DIMENSION(2) :: PLUSINFTY
        EQUIVALENCE (IEEE_PLUSINFTY,PLUSINFTY)
        DATA PLUSINFTY / Z'7FF00000',Z'00000000'/
        GET_IEEE_PLUSINFTY = IEEE_PLUSINFTY
    END FUNCTION GET_IEEE_PLUSINFTY

!*********************************************************************
!**                                                                   **
!**   FUNCTION: GET_IEEE_MINUSINFTY()                                 **
!**                                                                   **
!**   Input   : None.                                                 **
!**                                                                   **
!**   Output  : REAL, An IEEE representation of negative infinity.    **
!**                                                                   **
!*********************************************************************

    REAL(fpkind) FUNCTION GET_IEEE_MINUSINFTY()
        REAL(fpkind), SAVE :: IEEE_MINUSINFTY
        INTEGER, DIMENSION(2) :: MINUSINFTY
        EQUIVALENCE (IEEE_MINUSINFTY,MINUSINFTY)
        DATA MINUSINFTY / Z'FFF00000',Z'00000000'/
        GET_IEEE_MINUSINFTY = IEEE_MINUSINFTY
```

```
    END FUNCTION GET_IEEE_MINUSINFTY

!*********************************************************************
!**                                                                 **
!**   FUNCTION: GET_IEEE_NAN()                                      **
!**                                                                 **
!**   Input   : None.                                               **
!**                                                                 **
!**   Output  : REAL, An IEEE representation of NAN.                **
!**                                                                 **
!**   Notes   : There are multiple representable NAN's.             **
!**                                                                 **
!*********************************************************************

    REAL(fpkind) FUNCTION GET_IEEE_NAN()
        REAL(fpkind), SAVE :: IEEE_NAN
        INTEGER, DIMENSION(2) :: NAN
        EQUIVALENCE (IEEE_NAN,NAN)
        DATA NAN / Z'7FFFFFFF',Z'FFFFFFFF'/
        GET_IEEE_NAN = IEEE_NAN
    END FUNCTION GET_IEEE_NAN

!*********************************************************************
!**                                                                 **
!**   FUNCTION: POS_I(X)                                            **
!**                                                                 **
!**   Input   : INTERVAL                                            **
!**                                                                 **
!**   Output  : INTERVAL, The input interval, this is the monadic + **
!**                                                                 **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION POS_I(X)
        TYPE(INTERVAL), INTENT(IN) :: X
        POS_I = X
    END FUNCTION POS_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_ADD_I(X,Y)                                        **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : INTERVAL, The sum of two intervals                  **
!**                                                                 **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION I_ADD_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_ADD_I%INF = NEAREST(X%INF + Y%INF, DOWN)
        I_ADD_I%SUP = NEAREST(X%SUP + Y%SUP, UP)
    END FUNCTION I_ADD_I
```

```
!*********************************************************************
!**                                                                 **
!**   FUNCTION: NEG_I(X)                                            **
!**                                                                 **
!**   Input   : INTERVAL                                            **
!**                                                                 **
!**   Output  : INTERVAL, The negation of the input interval,       **
!**             this is the monadic -                               **
!**                                                                 **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION NEG_I(INTVAL)
        TYPE(INTERVAL), INTENT(IN) :: INTVAL
        NEG_I%INF = -INTVAL%SUP
        NEG_I%SUP = -INTVAL%INF
    END FUNCTION NEG_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_SUB_I(X,Y)                                        **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : INTERVAL, The difference of two intervals           **
!**                                                                 **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION I_SUB_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_SUB_I%INF = NEAREST(X%INF - Y%SUP, DOWN)
        I_SUB_I%SUP = NEAREST(X%SUP - Y%INF, UP)
    END FUNCTION I_SUB_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_MUL_I(X,Y)                                        **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : INTERVAL, The product of two intervals              **
!**                                                                 **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION I_MUL_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        REAL(fpkind)               :: II, IS, SI, SS
        II = X%INF * Y%INF
        IS = X%INF * Y%SUP
        SI = X%SUP * Y%INF
        SS = X%SUP * Y%SUP
```

```
        I_MUL_I%INF = NEAREST(MIN(II, IS, SI, SS), DOWN)
        I_MUL_I%SUP = NEAREST(MAX(II, IS, SI, SS), UP)
    END FUNCTION I_MUL_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_DIV_I(X,Y)                                        **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : INTERVAL, The quotient of two intervals             **
!**                                                                 **
!**   Notes   : If both the numerator and the denominator contain   **
!**             zero, the result  is -infinity to +infinity.        **
!**                                                                 **
!*********************************************************************

    TYPE(INTERVAL) FUNCTION I_DIV_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        REAL(fpkind)               :: II, IS, SI, SS
        IF (zero .IN. Y) THEN
            !
            ! We have divide by zero. Set the interval to plus and minus
            ! infinity and then check to see if one of the endpoints
            ! is zero.
            !
            I_DIV_I%INF = GET_IEEE_MINUSINFTY()
            I_DIV_I%SUP = GET_IEEE_PLUSINFTY()

            IF (ZERO .IN. X) THEN
                !
                ! Zero divided by zero
                !
                CALL SIGNAL_ERROR("Undefined division operation")
            ELSE IF ((Y%INF == zero) .AND. (Y%SUP > zero)) THEN
                !
                ! Denominator is non-negative
                !
                IF (zero < X%INF) THEN
                    !
                    ! positive / non-negative so the result is
                    ! <X%INF / Y%SUP, infinity>
                    !
                    I_DIV_I%INF = X%INF / Y%SUP
                ELSE IF (X%SUP < zero) THEN
                    !
                    ! negative / non-negative so the result is
                    ! <-infinity, X%SUP / Y%SUP>
                    !
                    I_DIV_I%SUP = X%SUP / Y%SUP
                END IF
```

```
            ELSE IF ((Y%SUP == zero) .AND. (Y%INF < zero)) THEN
                !
                ! Denominator is non-positive
                !
                IF (zero < X%INF) THEN
                    !
                    ! positive / non-positive so the result is
                    ! <-infinity, X%INF / Y%INF>
                    !
                    I_DIV_I%SUP = X%INF / Y%INF
                ELSE IF (X%SUP < zero) THEN
                    !
                    ! negative / non-positive so the result is
                    ! <X%SUP / Y%INF, infinity>
                    !
                    I_DIV_I%INF = X%SUP / Y%INF
                END IF
            END IF
        ELSE
            II = X%INF / Y%INF
            IS = X%INF / Y%SUP
            SI = X%SUP / Y%INF
            SS = X%SUP / Y%SUP
            I_DIV_I%INF = NEAREST(MIN(II, IS, SI, SS), DOWN)
            I_DIV_I%SUP = NEAREST(MAX(II, IS, SI, SS), UP)
        END  IF
    END FUNCTION I_DIV_I

!*****************************************************************
!**                                                             **
!**   FUNCTION: IVAL(X,[Y])                                     **
!**                                                             **
!**   Input   : One or two REALs                                **
!**                                                             **
!**   Output  : INTERVAL, The INTERVAL containing the number(s) **
!**             It is assumed that the left number is INF       **
!**                                                             **
!*****************************************************************

   TYPE(INTERVAL) FUNCTION IVAL(X, Y)
        REAL(fpkind), INTENT(IN)             :: X
        REAL(fpkind), INTENT(IN), OPTIONAL   :: Y
        IVAL%INF = X
        IVAL%SUP = X
        IF ( PRESENT(Y) ) THEN
            IF (X <= Y) THEN
                IVAL%SUP = Y
            ELSE
                CALL SIGNAL_ERROR("Reversed bounds in IVAL")
                IVAL%INF = Y
            END IF
        END IF
```

```
    END FUNCTION IVAL

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_LT_I(X,Y)                                         **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if every point in the first interval is less **
!**             than every point in the second interval, .FALSE.    **
!**             otherwise.                                          **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_LT_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_LT_I = X%SUP < Y%INF
    END FUNCTION I_LT_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_GT_I(X,Y)                                         **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if every point in the first interval is      **
!**             greater than every point in the second interval,    **
!**             .FALSE. otherwise.                                  **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_GT_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_GT_I = X%INF > Y%SUP
    END FUNCTION I_GT_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_LE_I(X,Y)                                         **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if every point in the first interval is      **
!**             less than or equal to every point in the second     **
!**             interval, .FALSE. otherwise.                        **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_LE_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_LE_I = X%SUP <= Y%INF
    END FUNCTION I_LE_I
```

```
!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_GE_I(X,Y)                                         **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if every point in the first interval is      **
!**             greater than or equal to every point in the second  **
!**             interval, .FALSE. otherwise.                        **
!**            interval that is less than some point in the        **
!**             second interval, .FALSE. otherwise.                 **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_PLT_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_PLT_I = X%INF < Y%SUP
    END FUNCTION I_PLT_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_PGT_I(X,Y)                                        **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if there exists a point in the first         **
!**             interval that is greater than some point in the     **
!**             second interval, .FALSE. otherwise.                 **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_PGT_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_PGT_I = X%SUP > Y%INF
    END FUNCTION I_PGT_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_PLE_I(X,Y)                                        **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if there exists a point in the first         **
!**             interval that is less than or equal to some point   **
!**             in the second interval, .FALSE. otherwise.          **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_PLE_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_PLE_I = X%INF <= Y%SUP
```

```
    END FUNCTION I_PLE_I

!**********************************************************************
!**                                                                  **
!**   FUNCTION: I_PGE_I(X,Y)                                         **
!**                                                                  **
!**   Input   : Two INTERVALs                                        **
!**                                                                  **
!**   Output  : .TRUE. if there exists a point in the first          **
!**             interval that is greater than or equal to some point**
!**             in the second interval, .FALSE. otherwise.           **
!**                                                                  **
!**********************************************************************

    LOGICAL FUNCTION I_PGE_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_PGE_I = X%SUP >= Y%INF
    END FUNCTION I_PGE_I

!**********************************************************************
!**                                                                  **
!**   FUNCTION: I_EQ_I(X,Y)                                          **
!**                                                                  **
!**   Input   : Two INTERVALs                                        **
!**                                                                  **
!**   Output  : .TRUE. if the intervals are the same,                **
!**             .FALSE. otherwise.                                   **
!**                                                                  **
!**********************************************************************

    LOGICAL FUNCTION I_EQ_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_EQ_I = X%INF == Y%INF .AND. X%SUP == Y%SUP
    END FUNCTION I_EQ_I

!**********************************************************************
!**                                                                  **
!**   FUNCTION: I_NE_I(X,Y)                                          **
!**                                                                  **
!**   Input   : Two INTERVALs                                        **
!**                                                                  **
!**   Output  : .TRUE. if the intervals are not the same,            **
!**             .FALSE. otherwise.                                   **
!**                                                                  **
!**********************************************************************

    LOGICAL FUNCTION I_NE_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_NE_I = X%INF /= Y%INF .OR. X%SUP /= Y%SUP
    END FUNCTION I_NE_I

!**********************************************************************
```

```
!**                                                                  **
!**   FUNCTION: I_INTERSECT_I(X,Y)                                   **
!**                                                                  **
!**   Input   : Two INTERVALs                                        **
!**                                                                  **
!**   Output  : INTERVAL, intersection of the two intervals, and     **
!**             [infinity, -infinity] if they do not intersect.      **
!**                                                                  **
!**********************************************************************

    TYPE(INTERVAL) FUNCTION I_INTERSECT_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        IF (X .DJ. Y) THEN
            I_INTERSECT_I%INF = GET_IEEE_PLUSINFTY()
            I_INTERSECT_I%SUP = GET_IEEE_MINUSINFTY()
            !
            ! Generating an error is currently not in the report
            !
            CALL SIGNAL_ERROR("Arguments to .IS. are disjoint")
        ELSE
            I_INTERSECT_I%INF = MAX(X%INF, Y%INF)
            I_INTERSECT_I%SUP = MIN(X%SUP, Y%SUP)
        END IF
    END FUNCTION I_INTERSECT_I

!**********************************************************************
!**                                                                  **
!**   FUNCTION: I_CONVEX_HULL_I(X,Y)                                 **
!**                                                                  **
!**   Input   : Two INTERVALs                                        **
!**                                                                  **
!**   Output  : INTERVAL, convex hull of the two intervals           **
!**                                                                  **
!**********************************************************************

    TYPE(INTERVAL) FUNCTION I_CONVEX_HULL_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_CONVEX_HULL_I%INF = MIN(X%INF, Y%INF)
        I_CONVEX_HULL_I%SUP = MAX(X%SUP, Y%SUP)
    END FUNCTION I_CONVEX_HULL_I

!**********************************************************************
!**                                                                  **
!**   FUNCTION: I_SUBSET_I(X,Y)                                      **
!**                                                                  **
!**   Input   : Two INTERVALs                                        **
!**                                                                  **
!**   Output  : .TRUE. if X is a subset of Y,                        **
!**             .FALSE. otherwise.                                   **
!**                                                                  **
!**********************************************************************
```

```
    LOGICAL FUNCTION I_SUBSET_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_SUBSET_I = X%INF >= Y%INF .AND. X%SUP <= Y%SUP
    END FUNCTION I_SUBSET_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_PROPER_SUBSET_I(X,Y)                              **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if X is a proper subset of Y,                **
!**             .FALSE. otherwise.                                  **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_PROPER_SUBSET_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_PROPER_SUBSET_I = (X .SB. Y) .AND.                    &
&                   X%INF > Y%INF .OR. X%SUP < Y%SUP
    END FUNCTION I_PROPER_SUBSET_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_SUPERSET_I(X,Y)                                   **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if the first is a superset of the second,    **
!**             .FALSE. otherwise.                                  **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_SUPERSET_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_SUPERSET_I = Y .SB. X
    END FUNCTION I_SUPERSET_I

!*********************************************************************
!**                                                                 **
!**   FUNCTION: I_PROPER_SUPERSET_I(X,Y)                            **
!**                                                                 **
!**   Input   : Two INTERVALs                                       **
!**                                                                 **
!**   Output  : .TRUE. if the first is a proper superset of the     **
!**             second, .FALSE. otherwise.                          **
!**                                                                 **
!*********************************************************************

    LOGICAL FUNCTION I_PROPER_SUPERSET_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_PROPER_SUPERSET_I = Y .PSB. X
```

```
    END FUNCTION I_PROPER_SUPERSET_I

!**********************************************************************
!**                                                                  **
!**   FUNCTION: I_DISJOINT_I(X,Y)                                    **
!**                                                                  **
!**   Input   : Two INTERVALs                                        **
!**                                                                  **
!**   Output  : .TRUE. if the two intervals are disjoint, that is,   **
!**             have no points in common, .FALSE. otherwise.         **
!**                                                                  **
!**********************************************************************

    LOGICAL FUNCTION I_DISJOINT_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_DISJOINT_I = X%INF > Y%SUP .OR. X%SUP < Y%INF
    END FUNCTION I_DISJOINT_I

!**********************************************************************
!**                                                                  **
!**   FUNCTION: R_IN_I(X,Y)                                          **
!**                                                                  **
!**   Input   : REAL and an INTERVAL                                 **
!**                                                                  **
!**   Output  : .TRUE. if the number is in the interval,             **
!**             .FALSE. otherwise.                                   **
!**                                                                  **
!**********************************************************************

    LOGICAL FUNCTION R_IN_I(X, Y)
        REAL(fpkind), INTENT(IN)   :: X
        TYPE(INTERVAL), INTENT(IN) :: Y
        R_IN_I = Y%INF <= X .AND. X <= Y%SUP
    END FUNCTION R_IN_I

!**********************************************************************
!**                                                                  **
!**   FUNCTION: I_IN_I(X,Y)                                          **
!**                                                                  **
!**   Input   : Two INTERVALs                                        **
!**                                                                  **
!**   Output  : .TRUE. if the first interval is contained in the     **
!**             second, .FALSE. otherwise.                           **
!**             This is the same as subset.                          **
!**                                                                  **
!**********************************************************************

    LOGICAL FUNCTION I_IN_I(X, Y)
        TYPE(INTERVAL), INTENT(IN) :: X, Y
        I_IN_I = X .SB. Y
    END FUNCTION I_IN_I
```

```
!**********************************************************************
!**                                                                  **
!**   SUBROUTINE: SIGNAL_ERROR(MSG)                                  **
!**                                                                  **
!**   Input   : A message to print                                   **
!**                                                                  **
!**********************************************************************

    SUBROUTINE SIGNAL_ERROR(MSG)
        CHARACTER(LEN=*), INTENT(IN) :: MSG
        WRITE(*,*) "ERROR:",MSG
    END SUBROUTINE SIGNAL_ERROR

END MODULE WG5BASIC
```