Section 13: Intrinsic procedures and modules

There are four classes of intrinsic procedures: inquiry functions, elemental functions, transformational functions, and subroutines. Intrinsic subroutines may be elemental.

There are three sets of standard intrinsic modules: a system environment module, modules to support exception handling and IEEE arithmetic, and a module to support interoperability with the C programming language. The later two sets of modules are described in sections 14 and 15, respectively.

13.1 Intrinsic functions

An **intrinsic function** is an inquiry function, an elemental intrinsic function, or a transformational function. An **inquiry function** is one whose result depends on the properties of its principal argument other than the value of this argument; in fact, the argument value may be undefined. Unless the description of an inquiry function states otherwise, its primary argument is permitted to be an unallocated allocatable or a pointer that is not associated. An **elemental intrinsic function** is one that is specified for scalar arguments, but may be applied to array arguments as described in 12.7. All other intrinsic functions are **transformational functions**; they almost all have one or more array-valued arguments or an array-valued result. All standard intrinsic functions are pure.

NOTE 13.1

Intrinsic subroutines are used for functionalities involving side effects.

Generic names of standard intrinsic functions are listed in 13.8. In most cases, generic functions accept arguments of more than one type and the type of the result is the same as the type of the arguments. **Specific names** of standard intrinsic functions with corresponding generic names are listed in 13.10.

If an intrinsic function is used as an actual argument to a procedure, its specific name shall be used and it may be referenced in the called procedure only with scalar arguments. If an intrinsic function does not have a specific name, it shall not be used as an actual argument (12.4.1.3).

13.2 Elemental intrinsic procedures

Elemental intrinsic procedures behave as described in 12.7.

13.3 Arguments to intrinsic procedures

All intrinsic procedures may be invoked with either positional arguments or argument keywords (12.4). The descriptions in 13.8 through 13.11 give the argument keyword names and positional sequence for standard intrinsic procedures.

Many of the intrinsic procedures have optional arguments. These arguments are identified by the notation "optional" in the argument descriptions. In addition, the names of the optional arguments are enclosed in square brackets in description headings and in lists of procedures. The valid forms of reference for procedures with optional arguments are described in 12.4.1.

NOTE 13.2

The text CMPLX (X [, Y, KIND]) indicates that Y and KIND are both optional arguments. Valid reference forms include CMPLX(*x*), CMPLX(*x, y*), CMPLX(*x*, KIND=*kind*), CMPLX(*x, y, kind*), and CMPLX(KIND=*kind*, X=*x*, Y=*y*).

NOTE 13.3

Some intrinsic procedures impose additional requirements on their optional arguments. For example, SELECTED_REAL_KIND requires that at least one of its optional arguments be present, and RANDOM_SEED requires that at most one of its optional arguments be present.

The dummy arguments of the specific intrinsic procedures in 13.10 have INTENT(IN). The dummy arguments of the generic intrinsic procedures in 13.11 have INTENT(IN) if the intent is not stated explicitly.

The actual argument associated with an intrinsic function dummy argument named KIND shall be a scalar integer initialization expression and its value shall specify a representation method for the function result that exists on the processor.

NOTE 13.4

Intrinsic procedures that assign values to arguments of type character do so in accordance with the rules of intrinsic assignment (7.5.1.4).

13.4 Bit manipulation and inquiry procedures

The bit manipulation procedures consist of a set of ten elemental functions and one elemental subroutine. Logical operations on bits are provided by the elemental functions IOR, IAND, NOT, and IEOR; shift operations are provided by the elemental functions ISHFT and ISHFTC; bit subfields may be referenced by the elemental function IBITS and by the elemental subroutine MVBITS; single-bit processing is provided by the elemental functions BTEST, IBSET, and IBCLR.

For the purposes of these procedures, a bit is defined to be a binary digit w located at position k of a nonnegative integer scalar object based on a model nonnegative integer defined by

$$j = \sum_{k=0}^{z-1} w_k \times 2^k$$

and for which w_k may have the value 0 or 1. An example of a model number compatible with the examples used in 13.5 would have $z = 32$, thereby defining a 32-bit integer.

An inquiry function BIT_SIZE is available to determine the parameter z of the model.

Effectively, this model defines an integer object to consist of z bits in sequence numbered from right to left from 0 to $z-1$. This model is valid only in the context of the use of such an object as the argument or result of one of the bit manipulation procedures. In all other contexts, the model defined for an integer in 13.5 applies. In particular, whereas the models are identical for $w_{z-1} = 0$, they do not correspond for $w_{z-1} = 1$ and the interpretation of bits in such objects is processor dependent.

13.5 Numeric manipulation and inquiry functions

The numeric manipulation and inquiry functions are described in terms of a model for the representation and behavior of numbers on a processor. The model has parameters that are determined so as to make the model best fit the machine on which the program is executed.

The model set for integer i is defined by

$$i = s \times \sum_{k=0}^{q-1} w_k \times r^k,$$

where r is an integer exceeding one, q is a positive integer, each w_k is a nonnegative integer less than r, and s is +1 or –1.

The model set for real x is defined by

$$x = \begin{cases} 0 \text{ or} \\ s \times b^e \times \sum_{k=1}^{p} f_k \times b^{-k}, \end{cases}$$

where b and p are integers exceeding one; each f_k is a nonnegative integer less than b, with f_1 nonzero; s is +1 or –1; and e is an integer that lies between some integer maximum $e_{\max}$ and some integer minimum $e_{\min}$ inclusively. For $x = 0$, its exponent e and digits f_k are defined to be zero. The integer parameters r and q determine the set of model integers and the integer parameters b, p, $e_{\min}$, and $e_{\max}$ determine the set of model floating point numbers. The parameters of the integer and real models are available for each integer and real data type implemented by the processor. The parameters characterize the set of available numbers in the definition of the model. The numeric manipulation and inquiry functions provide values related to the parameters and other constants related to them.

NOTE 13.5

Examples of these functions in this section use the models

$$i = s \times \sum_{k=0}^{30} w_k \times 2^k$$

and

$$x = 0 \text{ or } s \times 2^e \times \left(\frac{1}{2} + \sum_{k=2}^{24} f_k \times 2^{-k} \right), \quad -126 \le e \le 127$$

13.6 Array intrinsic functions

The array intrinsic functions perform the following operations on arrays: vector and matrix multiplication, numeric or logical computation that reduces the rank, array structure inquiry, array construction, array manipulation, and value location.

13.6.1 The shape of array arguments

The transformational array intrinsic functions operate on each array argument as a whole. The shape of the corresponding actual argument shall therefore be defined; that is, the actual argument shall be an array section, an assumed-shape array, an explicit-shape array, a pointer that is associated with a target, an allocatable array that has been allocated, or an array-valued expression. It shall not be an assumed-size array.

Some of the inquiry intrinsic functions accept array arguments for which the shape need not be defined. Assumed-size arrays may be used as arguments to these functions; they include the function LBOUND and certain references to SIZE and UBOUND.

13.6.2 Mask arguments

Some array intrinsic functions have an optional MASK argument that is used by the function to select the elements of one or more arguments to be operated on by the function. Any element not selected by the mask need not be defined at the time the function is invoked.

The MASK affects only the value of the function, and does not affect the evaluation, prior to invoking the function, of arguments that are array expressions.

A MASK argument shall be of type logical.

13.7 Standard intrinsic subroutines

Standard intrinsic subroutines are supplied by the processor and are defined in 13.9 and 13.11. An intrinsic subroutine is referenced by a CALL statement that uses its name explicitly. An intrinsic subroutine shall not be used as an actual argument. The effect of a subroutine reference is as specified in 13.11. The elemental subroutine MVBITS is pure. No other standard intrinsic subroutine is pure.

NOTE 13.6

As with user-written elemental subroutines, an elemental intrinsic subroutine is pure. The nonelemental intrinsic subroutines all have side effects (or reflect system side effects) and thus are not pure.

13.8 Standard generic intrinsic functions

For all of the standard intrinsic procedures, the arguments shown are the names that shall be used for argument keywords when using the keyword form for actual arguments.

NOTE 13.7

For example, a reference to CMPLX may be written in the form CMPLX (A, B, M) or in the form CMPLX (Y = B, KIND = M, X = A).

NOTE 13.8

Many of the argument keywords have names that are indicative of their usage. For example:

KIND	Describes the kind type parameter of the result
STRING, STRING_A	An arbitrary character string
BACK	Indicates a string scan is to be from right to left (backward)
MASK	A mask that may be applied to the arguments
DIM	A selected dimension of an array argument

13.8.1 Argument presence inquiry function

PRESENT (A)	Argument presence

13.8.2 Numeric functions

ABS (A)	Absolute value
AIMAG (Z)	Imaginary part of a complex number
AINT (A [, KIND])	Truncation to whole number
ANINT (A [, KIND])	Nearest whole number
CEILING (A [, KIND])	Least integer greater than or equal to number
CMPLX (X [, Y, KIND])	Conversion to complex type
CONJG (Z)	Conjugate of a complex number
DBLE (A)	Conversion to double precision real type
DIM (X, Y)	Positive difference
DPROD (X, Y)	Double precision real product
FLOOR (A [, KIND])	Greatest integer less than or equal to number
INT (A [, KIND])	Conversion to integer type
MAX (A1, A2 [, A3,...])	Maximum value
MIN (A1, A2 [, A3,...])	Minimum value
MOD (A, P)	Remainder function
MODULO (A, P)	Modulo function
NINT (A [, KIND])	Nearest integer

REAL (A [, KIND])	Conversion to real type
SIGN (A, B)	Transfer of sign

13.8.3 Mathematical functions

ACOS (X)	Arccosine
ASIN (X)	Arcsine
ATAN (X)	Arctangent
ATAN2 (Y, X)	Arctangent
COS (X)	Cosine
COSH (X)	Hyperbolic cosine
EXP (X)	Exponential
LOG (X)	Natural logarithm
LOG10 (X)	Common logarithm (base 10)
SIN (X)	Sine
SINH (X)	Hyperbolic sine
SQRT (X)	Square root
TAN (X)	Tangent
TANH (X)	Hyperbolic tangent

13.8.4 Character functions

ACHAR (I)	Character in given position in ASCII collating sequence
ADJUSTL (STRING)	Adjust left
ADJUSTR (STRING)	Adjust right
CHAR (I [, KIND])	Character in given position in processor collating sequence
IACHAR (C)	Position of a character in ASCII collating sequence
ICHAR (C)	Position of a character in processor collating sequence
INDEX (STRING, SUBSTRING [, BACK])	Starting position of a substring
LEN_TRIM (STRING)	Length without trailing blank characters
LGE (STRING_A, STRING_B)	Lexically greater than or equal
LGT (STRING_A, STRING_B)	Lexically greater than
LLE (STRING_A, STRING_B)	Lexically less than or equal
LLT (STRING_A, STRING_B)	Lexically less than
MAX (A1, A2 [, A3,...])	Maximum value
MIN (A1, A2 [, A3,...])	Minimum value
REPEAT (STRING, NCOPIES)	Repeated concatenation
SCAN (STRING, SET [, BACK])	Scan a string for a character in a set
TRIM (STRING)	Remove trailing blank characters
VERIFY (STRING, SET [, BACK])	Verify the set of characters in a string

13.8.5 Character inquiry function

LEN (STRING)	Length of a character entity

13.8.6 Kind functions

KIND (X)	Kind type parameter value
SELECTED_INT_KIND (R)	Integer kind type parameter value, given range
SELECTED_REAL_KIND ([P, R])	Real kind type parameter value,

	given precision and range
SELECTED_CHAR_KIND (NAME)	Character kind type parameter value, given character set name

13.8.7 Logical function

LOGICAL (L [, KIND])	Convert between objects of type logical with different kind type parameters

13.8.8 Numeric inquiry functions

DIGITS (X)	Number of significant digits of the model
EPSILON (X)	Number that is almost negligible compared to one
HUGE (X)	Largest number of the model
MAXEXPONENT (X)	Maximum exponent of the model
MINEXPONENT (X)	Minimum exponent of the model
PRECISION (X)	Decimal precision
RADIX (X)	Base of the model
RANGE (X)	Decimal exponent range
TINY (X)	Smallest positive number of the model

13.8.9 Bit inquiry function

BIT_SIZE (I)	Number of bits of the model

13.8.10 Bit manipulation functions

BTEST (I, POS)	Bit testing
IAND (I, J)	Bitwise AND
IBCLR (I, POS)	Clear bit
IBITS (I, POS, LEN)	Bit extraction
IBSET (I, POS)	Set bit
IEOR (I, J)	Exclusive OR
IOR (I, J)	Inclusive OR
ISHFT (I, SHIFT)	Logical shift
ISHFTC (I, SHIFT [, SIZE])	Circular shift
NOT (I)	Bitwise complement

13.8.11 Transfer function

TRANSFER (SOURCE, MOLD [, SIZE])	Treat first argument as if of type of second argument

13.8.12 Floating-point manipulation functions

EXPONENT (X)	Exponent part of a model number
FRACTION (X)	Fractional part of a number
NEAREST (X, S)	Nearest different processor number in given direction
RRSPACING (X)	Reciprocal of the relative spacing of model numbers near given number
SCALE (X, I)	Multiply a real by its base to an integer power
SET_EXPONENT (X, I)	Set exponent part of a number
SPACING (X)	Absolute spacing of model numbers near given number

13.8.13 Vector and matrix multiply functions

DOT_PRODUCT (VECTOR_A, VECTOR_B)	Dot product of two rank-one arrays
MATMUL (MATRIX_A, MATRIX_B)	Matrix multiplication

13.8.14 Array reduction functions

ALL (MASK [, DIM])	True if all values are true
ANY (MASK [, DIM])	True if any value is true
COUNT (MASK [, DIM])	Number of true elements in an array
MAXVAL (ARRAY, DIM [, MASK]) or MAXVAL (ARRAY [, MASK])	Maximum value in an array
MINVAL (ARRAY, DIM [, MASK]) or MINVAL (ARRAY [, MASK])	Minimum value in an array
PRODUCT (ARRAY, DIM [, MASK]) or PRODUCT (ARRAY [, MASK])	Product of array elements
SUM (ARRAY, DIM [, MASK]) or SUM (ARRAY [, MASK])	Sum of array elements

13.8.15 Array inquiry functions

LBOUND (ARRAY [, DIM])	Lower dimension bounds of an array
SHAPE (SOURCE)	Shape of an array or scalar
SIZE (ARRAY [, DIM])	Total number of elements in an array
UBOUND (ARRAY [, DIM])	Upper dimension bounds of an array

13.8.16 Array construction functions

MERGE (TSOURCE, FSOURCE, MASK)	Merge under mask
PACK (ARRAY, MASK [, VECTOR])	Pack an array into an array of rank one under a mask
SPREAD (SOURCE, DIM, NCOPIES)	Replicates array by adding a dimension
UNPACK (VECTOR, MASK, FIELD)	Unpack an array of rank one into an array under a mask

13.8.17 Array reshape function

RESHAPE (SOURCE, SHAPE[, PAD, ORDER])	Reshape an array

13.8.18 Array manipulation functions

CSHIFT (ARRAY, SHIFT [, DIM])	Circular shift
EOSHIFT (ARRAY, SHIFT [, BOUNDARY, DIM])	End-off shift
TRANSPOSE (MATRIX)	Transpose of an array of rank two

13.8.19 Array location functions

MAXLOC (ARRAY, DIM [, MASK]) or MAXLOC (ARRAY [, MASK])	Location of a maximum value in an array
MINLOC (ARRAY, DIM [, MASK]) or MINLOC (ARRAY [, MASK])	Location of a minimum value in an array

13.8.20 Allocation status inquiry functions

ALLOCATED (ARRAY) or ALLOCATED (SCALAR)	Allocation status

13.8.21 Pointer association status functions

ASSOCIATED (POINTER [, TARGET])	Association status inquiry or comparison
NULL ([MOLD])	Returns disassociated pointer

13.8.22 Type inquiry functions

EXTENDS_TYPE_OF (A, MOLD)	Same dynamic type or an extension
SAME_TYPE_AS (A, B)	Same dynamic type

13.8.23 System environment inquiry functions

COMMAND_ARGUMENT_COUNT ()	Number of command arguments

13.9 Standard intrinsic subroutines

CPU_TIME (TIME)	Obtain processor time
DATE_AND_TIME ([DATE, TIME, ZONE, VALUES])	Obtain date and time
GET_COMMAND ([COMMAND, LENGTH, STATUS])	Returns entire command
GET_COMMAND_ARGUMENT (NUMBER [, VALUE, LENGTH, STATUS])	Returns a command argument
GET_ENVIRONMENT_VARIABLE (NAME [, VALUE, LENGTH, TRIM_NAME])	Obtain the value of an environment variable
MVBITS (FROM, FROMPOS, LEN, TO, TOPOS)	Copies bits from one integer to another
RANDOM_NUMBER (HARVEST)	Returns pseudorandom number
RANDOM_SEED ([SIZE, PUT, GET])	Initializes or restarts the pseudorandom number generator
SYSTEM_CLOCK ([COUNT, COUNT_RATE, COUNT_MAX])	Obtain data from the system clock

13.10 Specific names for standard intrinsic functions

Specific Name	Generic Name	Argument Type
ABS (A)	ABS (A)	default real
ACOS (X)	ACOS (X)	default real
AIMAG (Z)	AIMAG (Z)	default complex
AINT (A)	AINT (A)	default real
ALOG (X)	LOG (X)	default real
ALOG10 (X)	LOG10 (X)	default real
• AMAX0 (A1, A2 [, A3,...])	REAL (MAX (A1, A2 [, A3,...]))	default integer
• AMAX1 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	default real
• AMIN0 (A1, A2 [, A3,...])	REAL (MIN (A1, A2 [, A3,...]))	default integer
• AMIN1 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	default real
AMOD (A, P)	MOD (A, P)	default real
ANINT (A)	ANINT (A)	default real
ASIN (X)	ASIN (X)	default real
ATAN (X)	ATAN (X)	default real
ATAN2 (Y, X)	ATAN2 (Y, X)	default real
CABS (A)	ABS (A)	default complex
CCOS (X)	COS (X)	default complex
CEXP (X)	EXP (X)	default complex

CHAR (I)	CHAR (I)	default integer
CLOG (X)	LOG (X)	default complex
CONJG (Z)	CONJG (Z)	default complex
COS (X)	COS (X)	default real
COSH (X)	COSH (X)	default real
CSIN (X)	SIN (X)	default complex
CSQRT (X)	SQRT (X)	default complex
DABS (A)	ABS (A)	double precision real
DACOS (X)	ACOS (X)	double precision real
DASIN (X)	ASIN (X)	double precision real
DATAN (X)	ATAN (X)	double precision real
DATAN2 (Y, X)	ATAN2 (Y, X)	double precision real
DCOS (X)	COS (X)	double precision real
DCOSH (X)	COSH (X)	double precision real
DDIM (X, Y)	DIM (X, Y)	double precision real
DEXP (X)	EXP (X)	double precision real
DIM (X, Y)	DIM (X, Y)	default real
DINT (A)	AINT (A)	double precision real
DLOG (X)	LOG (X)	double precision real
DLOG10 (X)	LOG10 (X)	double precision real
• DMAX1 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	double precision real
• DMIN1 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	double precision real
DMOD (A, P)	MOD (A, P)	double precision real
DNINT (A)	ANINT (A)	double precision real
DPROD (X, Y)	DPROD (X, Y)	default real
DSIGN (A, B)	SIGN (A, B)	double precision real
DSIN (X)	SIN (X)	double precision real
DSINH (X)	SINH (X)	double precision real
DSQRT (X)	SQRT (X)	double precision real
DTAN (X)	TAN (X)	double precision real
DTANH (X)	TANH (X)	double precision real
EXP (X)	EXP (X)	default real
• FLOAT (A)	REAL (A)	default integer
IABS (A)	ABS (A)	default integer
• ICHAR (C)	ICHAR (C)	default character
IDIM (X, Y)	DIM (X, Y)	default integer
• IDINT (A)	INT (A)	double precision real
IDNINT (A)	NINT (A)	double precision real
• IFIX (A)	INT (A)	default real
INDEX (STRING, SUBSTRING)	INDEX (STRING, SUBSTRING)	default character
• INT (A)	INT (A)	default real
ISIGN (A, B)	SIGN (A, B)	default integer
LEN (STRING)	LEN (STRING)	default character
• LGE (STRING_A, STRING_B)	LGE (STRING_A, STRING_B)	default character
• LGT (STRING_A, STRING_B)	LGT (STRING_A, STRING_B)	default character
• LLE (STRING_A, STRING_B)	LLE (STRING_A, STRING_B)	default character
• LLT (STRING_A, STRING_B)	LLT (STRING_A, STRING_B)	default character
• MAX0 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	default integer
• MAX1 (A1, A2 [, A3,...])	INT (MAX (A1, A2 [, A3,...]))	default real
• MIN0 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	default integer
• MIN1 (A1, A2 [, A3,...])	INT (MIN (A1, A2 [, A3,...]))	default real
MOD (A, P)	MOD (A, P)	default integer
NINT (A)	NINT (A)	default real
• REAL (A)	REAL (A)	default integer

SIGN (A, B)	SIGN (A, B)	default real
SIN (X)	SIN (X)	default real
SINH (X)	SINH (X)	default real
• SNGL (A)	REAL (A)	double precision real
SQRT (X)	SQRT (X)	default real
TAN (X)	TAN (X)	default real
TANH (X)	TANH (X)	default real

• These specific intrinsic functions shall not be used as an actual argument.

13.11 Specifications of the standard intrinsic procedures

This section contains detailed specifications of the standard generic intrinsic procedures in alphabetical order.

The types and type parameters of standard intrinsic procedure arguments and function results are determined by these specifications. The "Argument(s)" paragraphs specifiy requirements on the actual arguments of the procedures. The result characteristics are sometimes specified in terms of the characteristics of dummy arguments. A program is prohibited from invoking an intrinsic procedure under circumstances where a value to be returned in a subroutine argument or function result is outside the range of values representable by objects of the specified type and type parameters, unless the intrinsic module IEEE_ARITHMETIC (section 14) is accessible and there is support for an infinite or a NaN result, as appropriate. If an infinite result is returned, the flag IEEE_OVERFLOW or IEEE_DIVIDE_BY_ZERO shall signal; if a NaN result is returned, the flag IEEE_INVALID shall signal. If all results are normal, these flags shall have the same status as when the intrinsic procedure was invoked.

13.11.1 ABS (A)

Description. Absolute value.

Class. Elemental function.

Argument. A shall be of type integer, real, or complex.

Result Characteristics. The same as A except that if A is complex, the result is real.

Result Value. If A is of type integer or real, the value of the result is $|A|$; if A is complex with value (x, y), the result is equal to a processor-dependent approximation to $\sqrt{x^2 + y^2}$.

Example. ABS ((3.0, 4.0)) has the value 5.0 (approximately).

13.11.2 ACHAR (I)

Description. Returns the character in a specified position of the ASCII collating sequence. It is the inverse of the IACHAR function.

Class. Elemental function.

Argument. I shall be of type integer.

Result Characteristics. Default character of length one.

Result Value. If I has a value in the range $0 \le I \le 127$, the result is the character in position I of the ASCII collating sequence, provided the processor is capable of representing that character; otherwise, the result is processor dependent. ACHAR (IACHAR (C)) shall have the value C for any character C capable of representation in the processor.

Example. ACHAR (88) has the value 'X'.

13.11.3 ACOS (X)

Description. Arccosine (inverse cosine) function.

Class. Elemental function.

Argument. X shall be of type real with a value that satisfies the inequality $|X| \leq 1$.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to arccos(X), expressed in radians. It lies in the range $0 \leq \mathrm{ACOS}(X) \leq \pi$.

Example. ACOS (0.54030231) has the value 1.0 (approximately).

13.11.4 ADJUSTL (STRING)

Description. Adjust to the left, removing leading blanks and inserting trailing blanks.

Class. Elemental function.

Argument. STRING shall be of type character.

Result Characteristics. Character of the same length and kind type parameter as STRING.

Result Value. The value of the result is the same as STRING except that any leading blanks have been deleted and the same number of trailing blanks have been inserted.

Example. ADJUSTL (' WORD') has the value 'WORD '.

13.11.5 ADJUSTR (STRING)

Description. Adjust to the right, removing trailing blanks and inserting leading blanks.

Class. Elemental function.

Argument. STRING shall be of type character.

Result Characteristics. Character of the same length and kind type parameter as STRING.

Result Value. The value of the result is the same as STRING except that any trailing blanks have been deleted and the same number of leading blanks have been inserted.

Example. ADJUSTR ('WORD ') has the value ' WORD'.

13.11.6 AIMAG (Z)

Description. Imaginary part of a complex number.

Class. Elemental function.

Argument. Z shall be of type complex.

Result Characteristics. Real with the same kind type parameter as Z.

Result Value. If Z has the value (x, y), the result has value y.

Example. AIMAG ((2.0, 3.0)) has the value 3.0.

13.11.7 AINT (A [, KIND])

Description. Truncation to a whole number.

Class. Elemental function.

Arguments.

A	shall be of type real.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. The result is of type real. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of A.

Result Value. If $|A| < 1$, AINT (A) has the value 0; if $|A| \geq 1$, AINT (A) has a value equal to the integer whose magnitude is the largest integer that does not exceed the magnitude of A and whose sign is the same as the sign of A.

Examples. AINT (2.783) has the value 2.0. AINT (–2.783) has the value –2.0.

13.11.8 ALL (MASK [, DIM])

Description. Determine whether all values are true in MASK along dimension DIM.

Class. Transformational function.

Arguments.

MASK	shall be of type logical. It shall not be scalar.
DIM (optional)	shall be scalar and of type integer with value in the range $1 \leq \text{DIM} \leq n$, where n is the rank of MASK. The corresponding actual argument shall not be an optional dummy argument.

Result Characteristics. The result is of type logical with the same kind type parameter as MASK. It is scalar if DIM is absent or MASK has rank one; otherwise, the result is an array of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of MASK.

Result Value.

Case (i): The result of ALL (MASK) has the value true if all elements of MASK are true or if MASK has size zero, and the result has value false if any element of MASK is false.

Case (ii): If MASK has rank one, ALL (MASK, DIM) has a value equal to that of ALL (MASK). Otherwise, the value of element $(s_1, s_2, ..., s_{\text{DIM}-1}, s_{\text{DIM}+1}, ..., s_n)$ of ALL (MASK, DIM) is equal to ALL (MASK $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$).

Examples.

Case (i): The value of ALL ((/ .TRUE., .FALSE., .TRUE. /)) is false.

Case (ii): If B is the array $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$ and C is the array $\begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$ then ALL (B .NE. C, DIM = 1) is [true, false, false] and ALL (B .NE. C, DIM = 2) is [false, false].

13.11.9 ALLOCATED (ARRAY) or ALLOCATED (SCALAR)

Description. Indicate whether an allocatable variable is currently allocated.

Class. Inquiry function.

Arguments.

ARRAY	shall be an allocatable array.
SCALAR	shall be an allocatable scalar.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if the argument (ARRAY or SCALAR) is currently allocated and has the value false if the argument is not currently allocated.

13.11.10 ANINT (A [, KIND])

Description. Nearest whole number.

Class. Elemental function.

Arguments.

A	shall be of type real.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. The result is of type real. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of A.

Result Value. If $A > 0$, ANINT (A) has the value AINT (A + 0.5); if $A \leq 0$, ANINT (A) has the value AINT (A – 0.5).

Examples. ANINT (2.783) has the value 3.0. ANINT (–2.783) has the value –3.0.

13.11.11 ANY (MASK [, DIM])

Description. Determine whether any value is true in MASK along dimension DIM.

Class. Transformational function.

Arguments.

MASK	shall be of type logical. It shall not be scalar.
DIM (optional)	shall be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where n is the rank of MASK. The corresponding actual argument shall not be an optional dummy argument.

Result Characteristics. The result is of type logical with the same kind type parameter as MASK. It is scalar if DIM is absent or MASK has rank one; otherwise, the result is an array of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of MASK.

Result Value.

Case (i): The result of ANY (MASK) has the value true if any element of MASK is true and has the value false if no elements are true or if MASK has size zero.

Case (ii): If MASK has rank one, ANY (MASK, DIM) has a value equal to that of ANY (MASK). Otherwise, the value of element $(s_1, s_2, ..., s_{\text{DIM}-1}, s_{\text{DIM}+1}, ..., s_n)$ of ANY (MASK, DIM) is equal to ANY (MASK $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$).

Examples.

Case (i): The value of ANY ((/ .TRUE., .FALSE., .TRUE. /)) is true.

Case (ii): If B is the array $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$ and C is the array $\begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$ ANY (B .NE. C, DIM = 1) is [true, false, true] and ANY (B .NE. C, DIM = 2) is [true, true].

13.11.12 ASIN (X)

Description. Arcsine (inverse sine) function.

Class. Elemental function.

Argument. X shall be of type real. Its value shall satisfy the inequality $|X| \leq 1$.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to arcsin(X), expressed in radians. It lies in the range $-\pi/2 \leq \mathrm{ASIN}(X) \leq \pi/2$.

Example. ASIN (0.84147098) has the value 1.0 (approximately).

13.11.13 ASSOCIATED (POINTER [, TARGET])

Description. Returns the association status of its pointer argument or indicates whether the pointer is associated with the target.

Class. Inquiry function.

Arguments.

POINTER shall be a pointer. It may be of any type or may be a procedure pointer. Its pointer association status shall not be undefined.

TARGET (optional) shall be allowed as *target* in a pointer assignment statement (7.5.2) in which POINTER is *pointer-object*. If TARGET is a pointer then its pointer association status shall not be undefined.

Result Characteristics. Default logical scalar.

Result Value.

Case (i): If TARGET is absent, the result is true if POINTER is currently associated with a target and false if it is not.

Case (ii): If TARGET is present and is a procedure, the result is true if POINTER is associated with TARGET.

Case (iii): If TARGET is present and is a procedure pointer, the result is true if POINTER and TARGET are associated with the same procedure. If either POINTER or TARGET is disassociated, the result is false.

Case (iv): If TARGET is present and is a scalar target, the result is true if TARGET is not a zero-sized storage sequence and the target associated with POINTER occupies the same storage units as TARGET. Otherwise, the result is false. If the POINTER is disassociated, the result is false.

Case (v): If TARGET is present and is an array target, the result is true if the target associated with POINTER and TARGET have the same shape, are neither of size zero nor arrays whose elements are zero-sized storage sequences, and occupy the same storage units in array element order. Otherwise, the result is false. If POINTER is disassociated, the result is false.

Case (vi): If TARGET is present and is a scalar pointer, the result is true if the target associated with POINTER and the target associated with TARGET are not zero-sized storage sequences and they occupy the same storage units. Otherwise, the result is false. If either POINTER or TARGET is disassociated, the result is false.

Case (vii): If TARGET is present and is an array pointer, the result is true if the target associated with POINTER and the target associated with TARGET have the same shape, are neither of size zero nor arrays whose elements are zero-sized storage sequences, and occupy the same storage units in array element order. Otherwise, the result is false. If either POINTER or TARGET is disassociated, the result is false.

Examples. ASSOCIATED (CURRENT, HEAD) is true if CURRENT is associated with the target HEAD. After the execution of

```
A_PART => A (:N)
```

ASSOCIATED (A_PART, A) is true if N is equal to UBOUND (A, DIM = 1). After the execution of

```
NULLIFY (CUR); NULLIFY (TOP)
```

ASSOCIATED (CUR, TOP) is false.

13.11.14 ATAN (X)

Description. Arctangent (inverse tangent) function.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to arctan(X), expressed in radians, that lies in the range $-\pi/2 \le \mathrm{ATAN}(\mathrm{X}) \le \pi/2$.

Example. ATAN (1.5574077) has the value 1.0 (approximately).

13.11.15 ATAN2 (Y, X)

Description. Arctangent (inverse tangent) function. The result is the principal value of the argument of the nonzero complex number (X, Y).

Class. Elemental function.

Arguments.

Y	shall be of type real.
X	shall be of the same type and kind type parameter as Y. If Y has the value zero, X shall not have the value zero.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to the principal value of the argument of the complex number (X, Y), expressed in radians. It lies in the range $-\pi < ATAN2(Y, X) \le \pi$ and is equal to a processor-dependent approximation to a value of arctan(Y/X) if $\mathrm{X} \neq 0$. If $\mathrm{Y} > 0$, the result is positive. If Y = 0, the result is zero if $\mathrm{X} > 0$ and the result is π if $\mathrm{X} < 0$. If $\mathrm{Y} < 0$, the result is negative. If X = 0, the absolute value of the result is $\pi/2$.

Examples. ATAN2 (1.5574077, 1.0) has the value 1.0 (approximately). If Y has the value $\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$ and X has the value $\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$, the value of ATAN2 (Y, X) is approximately $\begin{bmatrix} \frac{3\pi}{4} & \frac{\pi}{4} \\ \frac{-3\pi}{4} & -\frac{\pi}{4} \end{bmatrix}$.

13.11.16 BIT_SIZE (I)

Description. Returns the number of bits z defined by the model of 13.4.

Class. Inquiry function.

Argument. I shall be of type integer. It may be scalar or array valued.

Result Characteristics. Scalar integer with the same kind type parameter as I.

Result Value. The result has the value of the number of bits z of the model integer defined for bit manipulation contexts in 13.4.

Example. BIT_SIZE (1) has the value 32 if z of the model is 32.

13.11.17 BTEST (I, POS)

Description. Tests a bit of an integer value.

Class. Elemental function.

Arguments.

I	shall be of type integer.
POS	shall be of type integer. It shall be nonnegative and be less than BIT_SIZE (I).

Result Characteristics. Default logical.

Result Value. The result has the value true if bit POS of I has the value 1 and has the value false if bit POS of I has the value 0. The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Examples. BTEST (8, 3) has the value true. If A has the value $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, the value of BTEST (A, 2) is $\begin{bmatrix} \text{false} & \text{false} \\ \text{false} & \text{true} \end{bmatrix}$ and the value of BTEST (2, A) is $\begin{bmatrix} \text{true} & \text{false} \\ \text{false} & \text{false} \end{bmatrix}$.

13.11.18 CEILING (A [, KIND])

Description. Returns the least integer greater than or equal to its argument.

Class. Elemental function.

Arguments.

A	shall be of type real.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default integer type.

Result Value. The result has a value equal to the least integer greater than or equal to A.

Examples. CEILING (3.7) has the value 4. CEILING (–3.7) has the value –3.

13.11.19 CHAR (I [, KIND])

Description. Returns the character in a given position of the processor collating sequence associated with the specified kind type parameter. It is the inverse of the ICHAR function.

Class. Elemental function.

Arguments.

I	shall be of type integer with a value in the range $0 \leq I \leq n-1$, where n is the number of characters in the collating sequence associated with the specified kind type parameter.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Character of length one. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default character type.

Result Value. The result is the character in position I of the collating sequence associated with the specified kind type parameter. ICHAR (CHAR (I, KIND (C))) shall have the value I for $0 \leq I \leq n-1$ and CHAR (ICHAR (C), KIND (C)) shall have the value C for any character C capable of representation in the processor.

Example. CHAR (88) has the value 'X' on a processor using the ASCII collating sequence.

13.11.20 CMPLX (X [, Y, KIND])

Description. Convert to complex type.

Class. Elemental function.

Arguments.

X	shall be of type integer, real, or complex.
Y (optional)	shall be of type integer or real. If X is of type complex, Y shall not be present, nor shall Y be associated with an optional dummy argument.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. The result is of type complex. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default real type.

Result Value. If Y is absent and X is not complex, it is as if Y were present with the value zero. If X is complex, it is as if Y were present with the value AIMAG (X). CMPLX (X, Y, KIND) has the complex value whose real part is REAL (X, KIND) and whose imaginary part is REAL (Y, KIND).

Example. CMPLX (–3) has the value (–3.0, 0.0).

13.11.21 COMMAND_ARGUMENT_COUNT ()

Description. Returns the number of command arguments.

Class. Inquiry function

Arguments. None.

Result Characteristics. Scalar default integer.

Result Value. The result value is equal to he number of command arguments available. If there are no command arguments available or if the processor does not support command arguments, then the result value is 0. If the processor has a concept of a command name, the command name does not count as one of the command arguments.

13.11.22 CONJG (Z)

Description. Conjugate of a complex number.

Class. Elemental function.

Argument. Z shall be of type complex.

Result Characteristics. Same as Z.

Result Value. If Z has the value (x, y), the result has the value $(x, -y)$.

Example. CONJG ((2.0, 3.0)) has the value (2.0, –3.0).

13.11.23 COS (X)

Description. Cosine function.

Class. Elemental function.

Argument. X shall be of type real or complex.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to cos(X). If X is of type real, it is regarded as a value in radians. If X is of type complex, its real part is regarded as a value in radians.

Example. COS (1.0) has the value 0.54030231 (approximately).

13.11.24 COSH (X)

Description. Hyperbolic cosine function.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to cosh(X).

Example. COSH (1.0) has the value 1.5430806 (approximately).

13.11.25 COUNT (MASK [, DIM, KIND])

Description. Count the number of true elements of MASK along dimension DIM.

Class. Transformational function.

Arguments.

MASK	shall be of type logical. It shall not be scalar.
DIM (optional)	shall be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where n is the rank of MASK. The corresponding actual argument shall not be an optional dummy argument.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type. The result is scalar if DIM is absent or MASK has rank one; otherwise, the result is an array of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of MASK.

Result Value.

Case (i): The result of COUNT (MASK) has a value equal to the number of true elements of MASK or has the value zero if MASK has size zero.

Case (ii): If MASK has rank one, COUNT (MASK, DIM) has a value equal to that of COUNT (MASK). Otherwise, the value of element $(s_1, s_2, ..., s_{\text{DIM}-1}, s_{\text{DIM}+1}, ..., s_n)$ of COUNT (MASK, DIM) is equal to COUNT (MASK $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$).

Examples.

Case (i): The value of COUNT ((/ .TRUE., .FALSE., .TRUE. /)) is 2.

Case (i): If B is the array $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$ and C is the array $\begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$, COUNT (B .NE. C, DIM = 1) is [2, 0, 1] and COUNT (B .NE. C, DIM = 2) is [1, 2].

13.11.26 CPU_TIME (TIME)

Description. Returns the processor time.

Class. Subroutine.

Argument. TIME shall be scalar and of type real. It is an INTENT(OUT) argument that is assigned a processor-dependent approximation to the processor time in seconds. If the processor cannot return a meaningful time, a processor-dependent negative value is returned.

Example.

```
REAL T1, T2
...
CALL CPU_TIME(T1)
...   ! Code to be timed.
CALL CPU_TIME(T2)
WRITE (*,*) 'Time taken by code was ', T2-T1, ' seconds'
```

writes the processor time taken by a piece of code.

NOTE 13.9

A processor for which a single result is inadequate (for example, a parallel processor) might choose to provide an additional version for which time is an array.

The exact definition of time is left imprecise because of the variability in what different processors are able to provide. The primary purpose is to compare different algorithms on the same computer or discover which parts of a calculation on a computer are the most expensive.

The start time is left imprecise because the purpose is to time sections of code, as in the example.

Most computer systems have multiple concepts of time. One common concept is that of time expended by the processor for a given program. This may or may not include system overhead, and has no obvious connection to elapsed "wall clock" time.

13.11.27 CSHIFT (ARRAY, SHIFT [, DIM])

Description. Perform a circular shift on an array expression of rank one or perform circular shifts on all the complete rank one sections along a given dimension of an array expression of rank two or greater. Elements shifted out at one end of a section are shifted in at the other end. Different sections may be shifted by different amounts and in different directions.

Class. Transformational function.

Arguments.

ARRAY	may be of any type. It shall not be scalar.
SHIFT	shall be of type integer and shall be scalar if ARRAY has rank one; otherwise, it shall be scalar or of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\mathrm{DIM}-1}, d_{\mathrm{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of ARRAY.
DIM (optional)	shall be a scalar and of type integer with a value in the range $1 \le \mathrm{DIM} \le n$, where n is the rank of ARRAY. If DIM is omitted, it is as if it were present with the value 1.

Result Characteristics. The result is of the type and type parameters of ARRAY, and has the shape of ARRAY.

Result Value.

Case (i): If ARRAY has rank one, element i of the result is ARRAY (1 + MODULO (i + SHIFT – 1, SIZE (ARRAY))).

Case (ii): If ARRAY has rank greater than one, section $(s_1, s_2, ..., s_{\mathrm{DIM}-1}, :, s_{\mathrm{DIM}+1}, ..., s_n)$ of the result has a value equal to CSHIFT (ARRAY $(s_1, s_2, ..., s_{\mathrm{DIM}-1}, :, s_{\mathrm{DIM}+1}, ..., s_n)$, sh, 1), where sh is SHIFT or SHIFT $(s_1, s_2, ..., s_{\mathrm{DIM}-1}, s_{\mathrm{DIM}+1}, ..., s_n)$.

Examples.

Case (i): If V is the array [1, 2, 3, 4, 5, 6], the effect of shifting V circularly to the left by two positions is achieved by CSHIFT (V, SHIFT = 2) which has the value [3, 4, 5, 6, 1, 2]; CSHIFT (V, SHIFT = –2) achieves a circular shift to the right by two positions and has the value [5, 6, 1, 2, 3, 4].

Case (ii): The rows of an array of rank two may all be shifted by the same amount or by different amounts. If M is the array $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, the value of CSHIFT (M, SHIFT = –1, DIM = 2) is $\begin{bmatrix} 3 & 1 & 2 \\ 6 & 4 & 5 \\ 9 & 7 & 8 \end{bmatrix}$, and the value of CSHIFT (M, SHIFT = (/ –1, 1, 0 /), DIM = 2) is $\begin{bmatrix} 3 & 1 & 2 \\ 5 & 6 & 4 \\ 7 & 8 & 9 \end{bmatrix}$.

13.11.28 DATE_AND_TIME ([DATE, TIME, ZONE, VALUES])

Description. Returns data about the real-time clock and date in a form compatible with the representations defined in ISO 8601:1988.

Class. Subroutine.

Arguments.

DATE (optional) shall be scalar and of type default character. It is an INTENT (OUT) argument. It is assigned a value of the form *CCYYMMDD*, where *CC* is the century, *YY* is the year within the century, *MM* is the month within the year, and *DD* is the day within the month. If there is no date available, DATE is assigned all blanks.

TIME (optional) shall be scalar and of type default character. It is an INTENT (OUT) argument. It is assigned a value of the form *hhmmss.sss*, where *hh* is the hour of the day, *mm* is the minutes of the hour, and *ss.sss* is the seconds and milliseconds of the minute. If there is no clock available, TIME is assigned all blanks.

ZONE (optional) shall be scalar and of type default character. It is an INTENT (OUT) argument. It is assigned a value of the form *+hhmm* or *–hhmm*, where *hh* and *mm* are the time difference with respect to Coordinated Universal Time (UTC) in hours and minutes, respectively. If this information is not available, ZONE is assigned all blanks.

VALUES (optional) shall be of type default integer and of rank one. It is an INTENT (OUT) argument. Its size shall be at least 8. The values returned in VALUES are as follows:

- VALUES (1) the year, including the century (for example, 1990), or –HUGE (0) if there is no date available;
- VALUES (2) the month of the year, or –HUGE (0) if there is no date available;
- VALUES (3) the day of the month, or –HUGE (0) if there is no date available;
- VALUES (4) the time difference with respect to Coordinated Universal Time (UTC) in minutes, or –HUGE (0) if this information is not available;
- VALUES (5) the hour of the day, in the range of 0 to 23, or –HUGE (0) if there is no clock;
- VALUES (6) the minutes of the hour, in the range 0 to 59, or –HUGE (0) if there is no clock;
- VALUES (7) the seconds of the minute, in the range 0 to 60, or –HUGE (0) if there is no clock;

VALUES (8) the milliseconds of the second, in the range 0 to 999, or –HUGE (0) if there is no clock.

Example.

```
INTEGER DATE_TIME (8)
CHARACTER (LEN = 10) BIG_BEN (3)
CALL DATE_AND_TIME (BIG_BEN (1), BIG_BEN (2),  &
                    BIG_BEN (3), DATE_TIME)
```

If run in Geneva, Switzerland on April 12, 1985 at 15:27:35.5 with a system configured for the local time zone, this sample would have assigned the value 19850412 to BIG_BEN (1), the value 152735.500 to BIG_BEN (2), and the value +0100 to BIG_BEN (3), and the following values to DATE_TIME: 1985, 4, 12, 60, 15, 27, 35, 500.

NOTE 13.10

UTC is defined by ISO 8601:1988.

13.11.29 DBLE (A)

Description. Convert to double precision real type.

Class. Elemental function.

Argument. A shall be of type integer, real, or complex.

Result Characteristics. Double precision real.

Result Value. The result has the value REAL (A, KIND (0.0D0)).

Example. DBLE (–3) has the value –3.0D0.

13.11.30 DIGITS (X)

Description. Returns the number of significant digits of the model representing numbers of the same type and kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type integer or real. It may be scalar or array valued.

Result Characteristics. Default integer scalar.

Result Value. The result has the value q if X is of type integer and p if X is of type real, where q and p are as defined in 13.5 for the model representing numbers of the same type and kind type parameter as X.

Example. DIGITS (X) has the value 24 for real X whose model is as at the end of 13.5.

13.11.31 DIM (X, Y)

Description. The difference X–Y if it is positive; otherwise zero.

Class. Elemental function.

Arguments.

X shall be of type integer or real.

Y shall be of the same type and kind type parameter as X.

Result Characteristics. Same as X.

Result Value. The value of the result is X–Y if X>Y and zero otherwise.

Example. DIM (–3.0, 2.0) has the value 0.0.

13.11.32 DOT_PRODUCT (VECTOR_A, VECTOR_B)

Description. Performs dot-product multiplication of numeric or logical vectors.

Class. Transformational function.

Arguments.

VECTOR_A shall be of numeric type (integer, real, or complex) or of logical type. It shall be array valued and of rank one.

VECTOR_B shall be of numeric type if VECTOR_A is of numeric type or of type logical if VECTOR_A is of type logical. It shall be array valued and of rank one. It shall be of the same size as VECTOR_A.

Result Characteristics. If the arguments are of numeric type, the type and kind type parameter of the result are those of the expression VECTOR_A * VECTOR_B determined by the types of the arguments according to 7.1.4. If the arguments are of type logical, the result is of type logical with the kind type parameter of the expression VECTOR_A **.**AND**.** VECTOR_B according to 7.1.4. The result is scalar.

Result Value.

Case (i): If VECTOR_A is of type integer or real, the result has the value SUM (VECTOR_A*VECTOR_B). If the vectors have size zero, the result has the value zero.

Case (ii): If VECTOR_A is of type complex, the result has the value SUM (CONJG (VECTOR_A)*VECTOR_B). If the vectors have size zero, the result has the value zero.

Case (iii): If VECTOR_A is of type logical, the result has the value ANY (VECTOR_A **.**AND**.** VECTOR_B). If the vectors have size zero, the result has the value false.

Example. DOT_PRODUCT ((/ 1, 2, 3 /), (/ 2, 3, 4 /)) has the value 20.

13.11.33 DPROD (X, Y)

Description. Double precision real product.

Class. Elemental function.

Arguments.

X shall be of type default real.

Y shall be of type default real.

Result Characteristics. Double precision real.

Result Value. The result has a value equal to a processor-dependent approximation to the product of X and Y.

Example. DPROD (–3.0, 2.0) has the value –6.0D0.

13.11.34 EOSHIFT (ARRAY, SHIFT [, BOUNDARY, DIM])

Description. Perform an end-off shift on an array expression of rank one or perform end-off shifts on all the complete rank-one sections along a given dimension of an array expression of rank two or greater. Elements are shifted off at one end of a section and copies of a boundary value are shifted in at the other end. Different sections may have different boundary values and may be shifted by different amounts and in different directions.

Class. Transformational function.

Arguments.

ARRAY may be of any type. It shall not be scalar.

SHIFT shall be of type integer and shall be scalar if ARRAY has rank one; otherwise, it shall be scalar or of rank $n-1$ and of shape $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of ARRAY.

BOUNDARY (optional) shall be of the same type and type parameters as ARRAY and shall be scalar if ARRAY has rank one; otherwise, it shall be either scalar or of rank $n-1$ and of shape $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n)$. BOUNDARY may be omitted for the data types in the following table and, in this case, it is as if it were present with the scalar value shown.

Type of ARRAY	Value of BOUNDARY
Integer	0
Real	0.0
Complex	(0.0, 0.0)
Logical	false
Character (*len*)	*len* blanks

DIM (optional) shall be scalar and of type integer with a value in the range $1 \le DIM \le n$, where n is the rank of ARRAY. If DIM is omitted, it is as if it were present with the value 1.

Result Characteristics. The result has the type, type parameters, and shape of ARRAY.

Result Value. Element $(s_1, s_2, ..., s_n)$ of the result has the value ARRAY $(s_1, s_2, ..., s_{DIM-1}, s_{DIM}+sh, s_{DIM+1}, ..., s_n)$ where sh is SHIFT or SHIFT $(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_n)$ provided the inequality LBOUND (ARRAY, DIM) $\le s_{DIM}+sh \le$ UBOUND (ARRAY, DIM) holds and is otherwise BOUNDARY or BOUNDARY $(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_n)$.

Examples.

Case (i): If V is the array [1, 2, 3, 4, 5, 6], the effect of shifting V end-off to the left by 3 positions is achieved by EOSHIFT (V, SHIFT = 3) which has the value [4, 5, 6, 0, 0, 0]; EOSHIFT (V, SHIFT = –2, BOUNDARY = 99) achieves an end-off shift to the right by 2 positions with the boundary value of 99 and has the value [99, 99, 1, 2, 3, 4].

Case (ii): The rows of an array of rank two may all be shifted by the same amount or by different amounts and the boundary elements can be the same or different. If M is the array $\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}$, then the value of

EOSHIFT (M, SHIFT = –1, BOUNDARY = '*', DIM = 2) is $\begin{bmatrix} * & A & B \\ * & D & E \\ * & G & H \end{bmatrix}$, and the value of EOSHIFT (M, SHIFT = (/ –1, 1, 0 /), BOUNDARY = (/ '*', '/', '?' /), DIM = 2) is $\begin{bmatrix} * & A & B \\ E & F & / \\ G & H & I \end{bmatrix}$.

13.11.35 EPSILON (X)

Description. Returns a positive model number that is almost negligible compared to unity of the model representing numbers of the same type and kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type real. It may be scalar or array valued.

Result Characteristics. Scalar of the same type and kind type parameter as X.

Result Value. The result has the value b^{1-p} where b and p are as defined in 13.5 for the model representing numbers of the same type and kind type parameter as X.

Example. EPSILON (X) has the value 2^{-23} for real X whose model is as at the end of 13.5.

13.11.36 EXP (X)

Description. Exponential.

Class. Elemental function.

Argument. X shall be of type real or complex.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to e^{X}. If X is of type complex, its imaginary part is regarded as a value in radians.

Example. EXP (1.0) has the value 2.7182818 (approximately).

13.11.37 EXPONENT (X)

Description. Returns the exponent part of the argument when represented as a model number.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Default integer.

Result Value. The result has a value equal to the exponent e of the model representation (13.5) for the value of X, provided X is nonzero and e is within the range for default integers. EXPONENT (X) has the value zero if X is zero.

Examples. EXPONENT (1.0) has the value 1 and EXPONENT (4.1) has the value 3 for reals whose model is as at the end of 13.5.

13.11.38 EXTENDS_TYPE_OF (A, MOLD)

Description. Inquires whether the dynamic type of A is an extension type (4.5.3) of the dynamic type of MOLD.

Class. Inquiry function.

Arguments.

A	shall be an object of extensible type. If it is a pointer, it shall not have an undefined association status.
MOLD	shall be an object of extensible type. If it is a pointer, it shall not have an undefined association status.

Result Characteristics. Default logical scalar.

Result Value. If MOLD is unlimited polymorphic and a disassociated pointer or unallocated allocatable, the result is true; otherwise if A is unlimited polymorphic and is a disassociated

pointer or an unallocated allocatable, the result is false; otherwise the result is true if and only if the dynamic type of A is an extension type of the dynamic type of MOLD.

NOTE 13.11

If either A or MOLD is a disassociated pointer, the dynamic type is the same as the declared type.

13.11.39 FLOOR (A [, KIND])

Description. Returns the greatest integer less than or equal to its argument.

Class. Elemental function.

Arguments.

A shall be of type real.

KIND (optional) shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default integer type.

Result Value. The result has value equal to the greatest integer less than or equal to A.

Examples. FLOOR (3.7) has the value 3. FLOOR (–3.7) has the value –4.

13.11.40 FRACTION (X)

Description. Returns the fractional part of the model representation of the argument value.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Same as X.

Result Value. The result has the value $X \times b^{-e}$, where b and e are as defined in 13.5 for the model representation of X. If X has the value zero, the result has the value zero.

Example. FRACTION (3.0) has the value 0.75 for reals whose model is as at the end of 13.5.

13.11.41 GET_COMMAND ([COMMAND, LENGTH, STATUS])

Description. Returns the entire command by which the program was invoked.

Class. Subroutine.

Arguments.

COMMAND (optional) shall be scalar and of type default character. It is an INTENT(OUT) argument. It is assigned the entire command by which the program was invoked. If the command cannot be determined, COMMAND is assigned all blanks.

LENGTH (optional) shall be scalar and of type default integer. It is an INTENT(OUT) argument. It is assigned the significant length of the command by which the program was invoked. The significant length may include trailing blanks if the processor allows commands with significant trailing blanks. This length does not consider any possible truncation or padding in assigning the command to the COMMAND argument; in fact the COMMAND argument need not even be present. If the command length cannot be determined, a length of 0 is assigned.

STATUS (optional) shall be scalar and of type default integer. It is an INTENT(OUT) argument. It is assigned the value 0 if the command retrieval is sucessful. It is assigned a processor-dependent non-zero value if the command retrieval fails.

13.11.42 GET_COMMAND_ARGUMENT (NUMBER [, VALUE, LENGTH, STATUS])

Description. Returns a command argument.

Class. Subroutine.

Arguments.

NUMBER — shall be scalar and of type default integer. It is an INTENT(IN) argument.

It specifies the number of the command argument that the other arguments give information about. Useful values of NUMBER are those between 0 and the argument count returned by the COMMAND_ARGUMENT_COUNT intrinsic. Other values are allowed, but will result in error status return (see below).

Command argument 0 is defined to be the command name by which the program was invoked if the processor has such a concept. It is allowed to call the GET_COMMAND_ARGUMENT procedure for command argument number 0, even if the processor does not define command names or other command arguments.

The remaining command arguments are numbered consecutively from 1 to the argument count in an order determined by the processor.

VALUE (optional) — shall be scalar and of type default character. It is an INTENT(OUT) argument. It is assigned the value of the command argument specified by NUMBER. If the command argument value cannot be determined, VALUE is assigned all blanks.

LENGTH (optional) — shall be scalar and of type default integer. It is an INTENT(OUT) argument. It is assigned the significant length of the command argument specified by NUMBER. The significant length may include trailing blanks if the processor allows command arguments with significant trailing blanks. This length does not consider any possible truncation or padding in assigning the command argument value to the VALUE argument; in fact the VALUE argument need not even be present. If the command argument length cannot be determined, a length of 0 is assigned.

STATUS (optional) — shall be scalar and of type default integer. It is an INTENT(OUT) argument. It is assigned the value 0 if the argument retrieval is sucessful. It is assigned a processor-dependent non-zero value if the argument retrieval fails.

NOTE 13.12

One possible reason for failure is that NUMBER is negative or greater than COMMAND_ARGUMENT_COUNT().

Example.

```
Program echo
  integer :: i
  character :: command*32, arg*128
  call get_command_argument(0, command)
  write (*,*) "Program name is: ", command
  do i = 1 , command_argument_count()
    call get_command_argument(i, arg)
    write (*,*) "Argument ", i, " is ", arg
  end do
end program echo
```

13.11.43 GET_ENVIRONMENT_VARIABLE (NAME [, VALUE, LENGTH, TRIM_NAME])

Description. Gets the value of an environment variable.

Class. Subroutine.

Arguments.

NAME (optional) shall be a scalar and of type default character. It is an INTENT(IN) argument.

VALUE (optional) shall be a scalar of type default character. It is an INTENT(OUT) argument. It is assigned the value of the environment variable specified by NAME. VALUE is assigned all blanks if the environment variable does not exist or does not have a value or if the processor does not support environment variables.

LENGTH (optional) shall be a scalar of type default integer. It is an INTENT(OUT) argument. LENGTH is set to either the length of the specified environment variable or to a negative value indicating an error. If the specified environment variable exists but has no value then LENGTH is set to zero. The value -1 indicates that the specified environment variable does not exist; -2 indicates that the processor does not support environment variables. The processor may return other negative values of LENGTH for other error conditions.

TRIM_NAME (optional)

shall be a scalar of type logical. It is an INTENT(IN) argument. If TRIM_NAME is present with the value false then trailing blanks in NAME are considered significant if the processor supports trailing blanks in environment variable names. Otherwise trailing blanks in NAME are not considered part of the environment variable's name.

13.11.44 HUGE (X)

Description. Returns the largest number of the model representing numbers of the same type and kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type integer or real. It may be scalar or array valued.

Result Characteristics. Scalar of the same type and kind type parameter as X.

Result Value. The result has the value $r^q - 1$ if X is of type integer and $(1 - b^{-p})b^{e_{\max}}$ if X is of type real, where r, q, b, p, and $e_{\max}$ are as defined in 13.5 for the model representing numbers of the same type and kind type parameter as X.

Example. HUGE (X) has the value $(1 - 2^{-24}) \times 2^{127}$ for real X whose model is as at the end of 13.5.

13.11.45 IACHAR (C)

Description. Returns the position of a character in the ASCII collating sequence. This is the inverse of the ACHAR function.

Class. Elemental function.

Argument. C shall be of type default character and of length one.

Result Characteristics. Default integer.

Result Value. If C is in the collating sequence defined by the codes specified in ISO/IEC 646:1991 (International Reference Version), the result is the position of C in that sequence and satisfies the inequality ($0 \leq \mathrm{IACHAR(C)} \leq 127$). A processor-dependent value is returned if

C is not in the ASCII collating sequence. The results are consistent with the LGE, LGT, LLE, and LLT lexical comparison functions. For example, if LLE (C, D) is true, IACHAR (C) **.**LE**.** IACHAR (D) is true where C and D are any two characters representable by the processor.

Example. IACHAR ('X') has the value 88.

13.11.46 IAND (I, J)

Description. Performs a bitwise AND.

Class. Elemental function.

Arguments.

I shall be of type integer.

J shall be of type integer with the same kind type parameter as I.

Result Characteristics. Same as I.

Result Value. The result has the value obtained by combining I and J bit-by-bit according to the following truth table:

I	J	IAND (I, J)
1	1	1
1	0	0
0	1	0
0	0	0

The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Example. IAND (1, 3) has the value 1.

13.11.47 IBCLR (I, POS)

Description. Clears one bit to zero.

Class. Elemental function.

Arguments.

I shall be of type integer.

POS shall be of type integer. It shall be nonnegative and less than BIT_SIZE (I).

Result Characteristics. Same as I.

Result Value. The result has the value of the sequence of bits of I, except that bit POS is zero. The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Examples. IBCLR (14, 1) has the result 12. If V has the value [1, 2, 3, 4], the value of IBCLR (POS = V, I = 31) is [29, 27, 23, 15].

13.11.48 IBITS (I, POS, LEN)

Description. Extracts a sequence of bits.

Class. Elemental function.

Arguments.

I shall be of type integer.

POS shall be of type integer. It shall be nonnegative and POS + LEN shall be less than or equal to BIT_SIZE (I).

LEN shall be of type integer and nonnegative.

Result Characteristics. Same as I.

Result Value. The result has the value of the sequence of LEN bits in I beginning at bit POS, right-adjusted and with all other bits zero. The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Example. IBITS (14, 1, 3) has the value 7.

13.11.49 IBSET (I, POS)

Description. Sets one bit to one.

Class. Elemental function.

Arguments.

I shall be of type integer.

POS shall be of type integer. It shall be nonnegative and less than BIT_SIZE (I).

Result Characteristics. Same as I.

Result Value. The result has the value of the sequence of bits of I, except that bit POS is one. The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Examples. IBSET (12, 1) has the value 14. If V has the value [1, 2, 3, 4], the value of IBSET (POS = V, I = 0) is [2, 4, 8, 16].

13.11.50 ICHAR (C)

Description. Returns the position of a character in the processor collating sequence associated with the kind type parameter of the character. This is the inverse of the CHAR function.

Class. Elemental function.

Argument. C shall be of type character and of length one. Its value shall be that of a character capable of representation in the processor.

Result Characteristics. Default integer.

Result Value. The result is the position of C in the processor collating sequence associated with the kind type parameter of C and is in the range $0 \leq \mathrm{ICHAR(C)} \leq n-1$, where n is the number of characters in the collating sequence. For any characters C and D capable of representation in the processor, C **.**LE**.** D is true if and only if ICHAR (C) **.**LE**.** ICHAR (D) is true and C **.**EQ**.** D is true if and only if ICHAR (C) **.**EQ**.** ICHAR (D) is true.

Example. ICHAR ('X') has the value 88 on a processor using the ASCII collating sequence for the default character type.

13.11.51 IEOR (I, J)

Description. Performs a bitwise exclusive OR.

Class. Elemental function.

Arguments.

I shall be of type integer.

J shall be of type integer with the same kind type parameter as I.

Result Characteristics. Same as I.

Result Value. The result has the value obtained by combining I and J bit-by-bit according to the following truth table:

I	J	IEOR (I, J)
1	1	0
1	0	1
0	1	1
0	0	0

The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Example. IEOR (1, 3) has the value 2.

13.11.52 INDEX (STRING, SUBSTRING [, BACK, KIND])

Description. Returns the starting position of a substring within a string.

Class. Elemental function.

Arguments.

STRING	shall be of type character.
SUBSTRING	shall be of type character with the same kind type parameter as STRING.
BACK (optional)	shall be of type logical.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type.

Result Value.

Case (i): If BACK is absent or has the value false, the result is the minimum positive value of I such that STRING (I : I + LEN (SUBSTRING) – 1) = SUBSTRING or zero if there is no such value. Zero is returned if LEN (STRING) < LEN (SUBSTRING) and one is returned if LEN (SUBSTRING) = 0.

Case (ii): If BACK is present with the value true, the result is the maximum value of I less than or equal to LEN (STRING) – LEN (SUBSTRING) + 1 such that STRING (I : I + LEN (SUBSTRING) – 1) = SUBSTRING or zero if there is no such value. Zero is returned if LEN (STRING) < LEN (SUBSTRING) and LEN (STRING) + 1 is returned if LEN (SUBSTRING) = 0.

Examples. INDEX ('FORTRAN', 'R') has the value 3.
INDEX ('FORTRAN', 'R', BACK = **.**TRUE**.**) has the value 5.

13.11.53 INT (A [, KIND])

Description. Convert to integer type.

Class. Elemental function.

Arguments.

A	shall be of type integer, real, or complex.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default integer type.

Result Value.

Case (i): If A is of type integer, INT (A) = A.

Case (ii): If A is of type real, there are two cases: if $|A| < 1$, INT (A) has the value 0; if $|A| \geq 1$, INT (A) is the integer whose magnitude is the largest integer that does not exceed the magnitude of A and whose sign is the same as the sign of A.

Case (iii): If A is of type complex, INT (A) is the value obtained by applying the case (ii) rule to the real part of A.

Example. INT (–3.7) has the value –3.

13.11.54 IOR (I, J)

Description. Performs a bitwise inclusive OR.

Class. Elemental function.

Arguments.

I shall be of type integer.

J shall be of type integer with the same kind type parameter as I.

Result Characteristics. Same as I.

Result Value. The result has the value obtained by combining I and J bit-by-bit according to the following truth table:

I	J	IOR (I, J)
1	1	1
1	0	1
0	1	1
0	0	0

The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Example. IOR (5, 3) has the value 7.

13.11.55 ISHFT (I, SHIFT)

Description. Performs a logical shift.

Class. Elemental function.

Arguments.

I shall be of type integer.

SHIFT shall be of type integer. The absolute value of SHIFT shall be less than or equal to BIT_SIZE (I).

Result Characteristics. Same as I.

Result Value. The result has the value obtained by shifting the bits of I by SHIFT positions. If SHIFT is positive, the shift is to the left; if SHIFT is negative, the shift is to the right; and if SHIFT is zero, no shift is performed. Bits shifted out from the left or from the right, as appropriate, are lost. Zeros are shifted in from the opposite end. The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Example. ISHFT (3, 1) has the result 6.

13.11.56 ISHFTC (I, SHIFT [, SIZE])

Description. Performs a circular shift of the rightmost bits.

Class. Elemental function.

Arguments.

I	shall be of type integer.
SHIFT	shall be of type integer. The absolute value of SHIFT shall be less than or equal to SIZE.
SIZE (optional)	shall be of type integer. The value of SIZE shall be positive and shall not exceed BIT_SIZE (I). If SIZE is absent, it is as if it were present with the value of BIT_SIZE (I).

Result Characteristics. Same as I.

Result Value. The result has the value obtained by shifting the SIZE rightmost bits of I circularly by SHIFT positions. If SHIFT is positive, the shift is to the left; if SHIFT is negative, the shift is to the right; and if SHIFT is zero, no shift is performed. No bits are lost. The unshifted bits are unaltered. The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Example. ISHFTC (3, 2, 3) has the value 5.

13.11.57 KIND (X)

Description. Returns the value of the kind type parameter of X.

Class. Inquiry function.

Argument. X may be of any intrinsic type. It may be scalar or array valued.

Result Characteristics. Default integer scalar.

Result Value. The result has a value equal to the kind type parameter value of X.

Example. KIND (0.0) has the kind type parameter value of default real.

13.11.58 LBOUND (ARRAY [, DIM, KIND])

Description. Returns all the lower bounds or a specified lower bound of an array.

Class. Inquiry function.

Arguments.

ARRAY	may be of any type. It shall not be scalar. It shall not be an unallocated allocatable or a pointer that is not associated.
DIM (optional)	shall be scalar and of type integer with a value in the range $1 \leq \mathrm{DIM} \leq n$, where n is the rank of ARRAY. The corresponding actual argument shall not be an optional dummy argument.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type. The result is scalar if DIM is present; otherwise, the result is an array of rank one and size n, where n is the rank of ARRAY.

Result Value.

Case (i): For an array section or for an array expression other than a whole array or array structure component, LBOUND (ARRAY, DIM) has the value 1. For a whole array or array structure component, LBOUND (ARRAY, DIM) has the value:

(a) equal to the lower bound for subscript DIM of ARRAY if dimension DIM of ARRAY does not have extent zero or if ARRAY is an assumed-size array of rank DIM, or

(b) 1 otherwise.

Case (ii): LBOUND (ARRAY) has a value whose *i*th component is equal to LBOUND (ARRAY, *i*), for *i* = 1, 2, ..., *n*, where *n* is the rank of ARRAY.

Examples. If A is declared by the statement

```
REAL A (2:3, 7:10)
```

then LBOUND (A) is [2, 7] and LBOUND (A, DIM=2) is 7.

13.11.59 LEN (STRING [, KIND])

Description. Returns the length of a character entity.

Class. Inquiry function.

Arguments.

STRING	shall be of type character. It may be scalar or array valued. If it is an unallocated allocatable or a pointer that is not associated, its length type parameter shall not be deferred.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer scalar. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type.

Result Value. The result has a value equal to the number of characters in STRING if it is scalar or in an element of STRING if it is array valued.

Example. If C is declared by the statement

```
CHARACTER (11) C (100)
```

LEN (C) has the value 11.

13.11.60 LEN_TRIM (STRING [, KIND])

Description. Returns the length of the character argument without counting trailing blank characters.

Class. Elemental function.

Arguments.

STRING	shall be of type character.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type.

Result Value. The result has a value equal to the number of characters remaining after any trailing blanks in STRING are removed. If the argument contains no nonblank characters, the result is zero.

Examples. LEN_TRIM (' A B ') has the value 4 and LEN_TRIM (' ') has the value 0.

13.11.61 LGE (STRING_A, STRING_B)

Description. Test whether a string is lexically greater than or equal to another string, based on the ASCII collating sequence.

Class. Elemental function.

Arguments.

STRING_A	shall be of type default character.
STRING_B	shall be of type default character.

Result Characteristics. Default logical.

Result Value. If the strings are of unequal length, the comparison is made as if the shorter string were extended on the right with blanks to the length of the longer string. If either string contains a character not in the ASCII character set, the result is processor dependent. The result is true if the strings are equal or if STRING_A follows STRING_B in the ASCII collating sequence; otherwise, the result is false.

NOTE 13.13

The result is true if both STRING_A and STRING_B are of zero length.

Example. LGE ('ONE', 'TWO') has the value false.

13.11.62 LGT (STRING_A, STRING_B)

Description. Test whether a string is lexically greater than another string, based on the ASCII collating sequence.

Class. Elemental function.

Arguments.

STRING_A	shall be of type default character.
STRING_B	shall be of type default character.

Result Characteristics. Default logical.

Result Value. If the strings are of unequal length, the comparison is made as if the shorter string were extended on the right with blanks to the length of the longer string. If either string contains a character not in the ASCII character set, the result is processor dependent. The result is true if STRING_A follows STRING_B in the ASCII collating sequence; otherwise, the result is false.

NOTE 13.14

The result is false if both STRING_A and STRING_B are of zero length.

Example. LGT ('ONE', 'TWO') has the value false.

13.11.63 LLE (STRING_A, STRING_B)

Description. Test whether a string is lexically less than or equal to another string, based on the ASCII collating sequence.

Class. Elemental function.

Arguments.

STRING_A	shall be of type default character.
STRING_B	shall be of type default character.

Result Characteristics. Default logical.

Result Value. If the strings are of unequal length, the comparison is made as if the shorter string were extended on the right with blanks to the length of the longer string. If either string contains a character not in the ASCII character set, the result is processor dependent. The result is true if the strings are equal or if STRING_A precedes STRING_B in the ASCII collating sequence; otherwise, the result is false.

NOTE 13.15

The result is true if both STRING_A and STRING_B are of zero length.

Example. LLE ('ONE', 'TWO') has the value true.

13.11.64 LLT (STRING_A, STRING_B)

Description. Test whether a string is lexically less than another string, based on the ASCII collating sequence.

Class. Elemental function.

Arguments.

STRING_A	shall be of type default character.
STRING_B	shall be of type default character.

Result Characteristics. Default logical.

Result Value. If the strings are of unequal length, the comparison is made as if the shorter string were extended on the right with blanks to the length of the longer string. If either string contains a character not in the ASCII character set, the result is processor dependent. The result is true if STRING_A precedes STRING_B in the ASCII collating sequence; otherwise, the result is false.

NOTE 13.16

The result is false if both STRING_A and STRING_B are of zero length.

Example. LLT ('ONE', 'TWO') has the value true.

13.11.65 LOG (X)

Description. Natural logarithm.

Class. Elemental function.

Argument. X shall be of type real or complex. If X is real, its value shall be greater than zero. If X is complex, its value shall not be zero.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to $\log_e X$. A result of type complex is the principal value with imaginary part ω in the range $-\pi < \omega \leq \pi$. The imaginary part of the result is π only when the real part of the argument is less than zero and the imaginary part of the argument is zero.

Example. LOG (10.0) has the value 2.3025851 (approximately).

13.11.66 LOG10 (X)

Description. Common logarithm.

Class. Elemental function.

Argument. X shall be of type real. The value of X shall be greater than zero.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to $\log_{10} X$.

Example. LOG10 (10.0) has the value 1.0 (approximately).

13.11.67 LOGICAL (L [, KIND])

Description. Converts between kinds of logical.

Class. Elemental function.

Arguments.

L	shall be of type logical.

KIND (optional) shall be a scalar integer initialization expression.

Result Characteristics. Logical. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default logical.

Result Value. The value is that of L.

Example. LOGICAL (L **.**OR**. .**NOT**.** L) has the value true and is of type default logical, regardless of the kind type parameter of the logical variable L.

13.11.68 MATMUL (MATRIX_A, MATRIX_B)

Description. Performs matrix multiplication of numeric or logical matrices.

Class. Transformational function.

Arguments.

MATRIX_A shall be of numeric type (integer, real, or complex) or of logical type. It shall be array valued and of rank one or two.

MATRIX_B shall be of numeric type if MATRIX_A is of numeric type and of logical type if MATRIX_A is of logical type. It shall be array valued and of rank one or two. If MATRIX_A has rank one, MATRIX_B shall have rank two. If MATRIX_B has rank one, MATRIX_A shall have rank two. The size of the first (or only) dimension of MATRIX_B shall equal the size of the last (or only) dimension of MATRIX_A.

Result Characteristics. If the arguments are of numeric type, the type and kind type parameter of the result are determined by the types of the arguments according to 7.1.4.2. If the arguments are of type logical, the result is of type logical with the kind type parameter of the arguments according to 7.1.4.2. The shape of the result depends on the shapes of the arguments as follows:

Case (i): If MATRIX_A has shape (n, m) and MATRIX_B has shape (m, k), the result has shape (n, k).

Case (ii): If MATRIX_A has shape (m) and MATRIX_B has shape (m, k), the result has shape (k).

Case (iii): If MATRIX_A has shape (n, m) and MATRIX_B has shape (m), the result has shape (n).

Result Value.

Case (i): Element (i, j) of the result has the value SUM (MATRIX_A $(i, :) *$ MATRIX_B $(:, j)$) if the arguments are of numeric type and has the value ANY (MATRIX_A $(i, :)$ **.**AND**.** MATRIX_B $(:, j)$) if the arguments are of logical type.

Case (ii): Element (j) of the result has the value SUM (MATRIX_A $(:) *$ MATRIX_B $(:, j)$) if the arguments are of numeric type and has the value ANY (MATRIX_A $(:)$ **.**AND**.** MATRIX_B $(:, j)$) if the arguments are of logical type.

Case (iii): Element (i) of the result has the value SUM (MATRIX_A $(i, :) *$ MATRIX_B $(:)$) if the arguments are of numeric type and has the value ANY (MATRIX_A $(i, :)$ **.**AND**.** MATRIX_B $(:)$) if the arguments are of logical type.

Examples. Let A and B be the matrices $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix}$ and $\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}$; let X and Y be the vectors [1, 2] and [1, 2, 3].

Case (i): The result of MATMUL (A, B) is the matrix-matrix product AB with the value $\begin{bmatrix} 14 & 20 \\ 20 & 29 \end{bmatrix}$.

Case (ii): The result of MATMUL (X, A) is the vector-matrix product XA with the value [5, 8, 11].

Case (iii): The result of MATMUL (A, Y) is the matrix-vector product AY with the value [14, 20].

13.11.69 MAX (A1, A2 [, A3, ...])

Description. Maximum value.

Class. Elemental function.

Arguments. The arguments shall all have the same type which shall be integer, real, or character and they shall all have the same kind type parameter.

Result Characteristics. The type and kind type parameter of the result are the same as those of the arguments. For arguments of character type, the length of the result is the length of the longest argument.

Result Value. The value of the result is that of the largest argument. For arguments of character type, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the arguments is applied. If the selected argument is shorter than the longest argument, the result is padded with blanks on the right to the length of the longest argument.

Examples. MAX (–9.0, 7.0, 2.0) has the value 7.0, MAX ("Z", "BB") has the value "Z ", and MAX ((/"A", "Z"/), (/"BB", "Y "/)) has the value (/"BB", "Z "/).

13.11.70 MAXEXPONENT (X)

Description. Returns the maximum exponent of the model representing numbers of the same type and kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type real. It may be scalar or array valued.

Result Characteristics. Default integer scalar.

Result Value. The result has the value $e_{\max}$, as defined in 13.5 for the model representing numbers of the same type and kind type parameter as X.

Example. MAXEXPONENT (X) has the value 127 for real X whose model is as at the end of 13.5.

13.11.71 MAXLOC (ARRAY, DIM [, MASK,KIND]) or MAXLOC (ARRAY [, MASK,KIND])

Description. Determine the location of the first element of ARRAY along dimension DIM having the maximum value of the elements identified by MASK.

Class. Transformational function.

Arguments.

ARRAY	shall be of type integer, real, or character. It shall not be scalar.
DIM	shall be scalar and of type integer with a value in the range $1 \le \text{DIM} \le n$, where n is the rank of ARRAY. The corresponding actual argument shall not be an optional dummy argument.
MASK (optional)	shall be of type logical and shall be conformable with ARRAY.

KIND (optional) shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type. If DIM is absent, the result is an array of rank one and of size equal to the rank of ARRAY; otherwise, the result is of rank $n-1$ and shape $(d_1, d_2, ..., d_{\mathrm{DIM}-1}, d_{\mathrm{DIM}+1}, ..., d_n)$, where $(d_1, d_2, ..., d_n)$ is the shape of ARRAY.

Result Value.

Case (i): The result of MAXLOC (ARRAY) is a rank-one array whose element values are the values of the subscripts of an element of ARRAY whose value equals the maximum value of all of the elements of ARRAY. The *i*th subscript returned lies in the range 1 to e_i, where e_i is the extent of the *i*th dimension of ARRAY. If more than one element has the maximum value, the element whose subscripts are returned is the first such element, taken in array element order. If ARRAY has size zero, the value of the result is processor dependent.

Case (ii): The result of MAXLOC (ARRAY, MASK = MASK) is a rank-one array whose element values are the values of the subscripts of an element of ARRAY, corresponding to a true element of MASK, whose value equals the maximum value of all such elements of ARRAY. The *i*th subscript returned lies in the range 1 to e_i, where e_i is the extent of the *i*th dimension of ARRAY. If more than one such element has the maximum value, the element whose subscripts are returned is the first such element taken in array element order. If ARRAY has size zero or every element of MASK has the value false, the value of the result is processor dependent.

Case (iii): If ARRAY has rank one, MAXLOC (ARRAY, DIM = DIM [, MASK = MASK]) is a scalar whose value is equal to that of the first element of MAXLOC (ARRAY [, MASK = MASK]). Otherwise, the value of element $(s_1, s_2, ..., s_{\mathrm{DIM}-1}, s_{\mathrm{DIM}+1}, ..., s_n)$ of the result is equal to

MAXLOC (ARRAY $(s_1, s_2, ..., s_{\mathrm{DIM}-1}, :, s_{\mathrm{DIM}+1}, ..., s_n)$, DIM=1
[, MASK = MASK $(s_1, s_2, ..., s_{\mathrm{DIM}-1}, :, s_{\mathrm{DIM}+1}, ..., s_n)$]).

If ARRAY has type character, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the arguments is applied.

Examples.

Case (i): The value of MAXLOC ((/ 2, 6, 4, 6 /)) is [2].

Case (ii): If A has the value $\begin{bmatrix} 0 & -5 & 8 & -3 \\ 3 & 4 & -1 & 2 \\ 1 & 5 & 6 & -4 \end{bmatrix}$, MAXLOC (A, MASK = A **.**LT**.** 6) has the value [3, 2]. Note that this is true even if A has a declared lower bound other than 1.

Case (iii): The value of MAXLOC ((/ 5, -9, 3 /), DIM = 1) is 1. If B has the value $\begin{bmatrix} 1 & 3 & -9 \\ 2 & 2 & 6 \end{bmatrix}$, MAXLOC (B, DIM = 1) is [2, 1, 2] and MAXLOC (B, DIM = 2) is [2, 3]. Note that this is true even if B has a declared lower bound other than 1.

13.11.72 MAXVAL (ARRAY, DIM [, MASK]) or MAXVAL (ARRAY [, MASK])

Description. Maximum value of the elements of ARRAY along dimension DIM corresponding to the true elements of MASK.

Class. Transformational function.

Arguments.

ARRAY	shall be of type integer, real, or character. It shall not be scalar.
DIM	shall be scalar and of type integer with a value in the range $1 \le \text{DIM} \le n$, where n is the rank of ARRAY. The corresponding actual argument shall not be an optional dummy argument.
MASK (optional)	shall be of type logical and shall be conformable with ARRAY.

Result Characteristics. The result is of the same type and type parameters as ARRAY. It is scalar if DIM is absent or ARRAY has rank one; otherwise, the result is an array of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of ARRAY.

Result Value.

Case (i): The result of MAXVAL (ARRAY) has a value equal to the maximum value of all the elements of ARRAY if the size of ARRAY is not zero . If ARRAY has size zero and type integer or real, the result has the value of the negative number of the largest magnitude supported by the processor for numbers of the type and kind type parameter of ARRAY. If ARRAY has size zero and type character, the result has the value of a string of characters of length LEN (ARRAY), with each character equal to CHAR (0, KIND = KIND (ARRAY)).

Case (ii): The result of MAXVAL (ARRAY, MASK = MASK) has a value equal to that of MAXVAL (PACK (ARRAY, MASK).

Case (iii): The result of MAXVAL (ARRAY, DIM = DIM [,MASK = MASK]) has a value equal to that of MAXVAL (ARRAY [,MASK = MASK]) if ARRAY has rank one. Otherwise, the value of element $(s_1, s_2, ..., s_{\text{DIM}-1}, s_{\text{DIM}+1}, ..., s_n)$ of the result is equal to

MAXVAL (ARRAY $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$
[, MASK = MASK $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$]).

If ARRAY has type character, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the arguments is applied.

Examples.

Case (i): The value of MAXVAL ((/ 1, 2, 3 /)) is 3.

Case (ii): MAXVAL (C, MASK = C **.**LT**.** 0.0) finds the maximum of the negative elements of C.

Case (iii): If B is the array $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, MAXVAL (B, DIM = 1) is [2, 4, 6] and MAXVAL (B, DIM = 2) is [5, 6].

13.11.73 MERGE (TSOURCE, FSOURCE, MASK)

Description. Choose alternative value according to the value of a mask.

Class. Elemental function.

Arguments.

TSOURCE	may be of any type.
FSOURCE	shall be of the same type and type parameters as TSOURCE.
MASK	shall be of type logical.

Result Characteristics. Same as TSOURCE.

Result Value. The result is TSOURCE if MASK is true and FSOURCE otherwise.

Examples. If TSOURCE is the array $\begin{bmatrix} 1 & 6 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, FSOURCE is the array $\begin{bmatrix} 0 & 3 & 2 \\ 7 & 4 & 8 \end{bmatrix}$ and MASK is the array $\begin{bmatrix} T & . & T \\ . & . & T \end{bmatrix}$, where "T" represents true and "." represents false, then MERGE (TSOURCE, FSOURCE, MASK) is $\begin{bmatrix} 1 & 3 & 5 \\ 7 & 4 & 6 \end{bmatrix}$. The value of MERGE (1.0, 0.0, K > 0) is 1.0 for K = 5 and 0.0 for K = –2.

13.11.74 MIN (A1, A2 [, A3, ...])

Description. Minimum value.

Class. Elemental function.

Arguments. The arguments shall all be of the same type which shall be integer, real, or character and they shall all have the same kind type parameter.

Result Characteristics. The type and kind type parameter of the result are the same as those of the arguments. For arguments of character type, the length of the result is the length of the longest argument.

Result Value. The value of the result is that of the smallest argument. For arguments of character type, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the arguments is applied. If the selected argument is shorter than the longest argument, the result is padded with blanks on the right to the length of the longest argument.

Example. MIN (–9.0, 7.0, 2.0) has the value –9.0, MIN ("A", "YY") has the value "A ", and MIN ((/"Z", "A"/), (/"YY", "B "/)) has the value (/"YY", "A "/).

13.11.75 MINEXPONENT (X)

Description. Returns the minimum (most negative) exponent of the model representing numbers of the same type and kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type real. It may be scalar or array valued.

Result Characteristics. Default integer scalar.

Result Value. The result has the value $e_{\min}$, as defined in 13.5 for the model representing numbers of the same type and kind type parameter as X.

Example. MINEXPONENT (X) has the value –126 for real X whose model is as at the end of 13.5.

13.11.76 MINLOC (ARRAY, DIM [, MASK,KIND]) or MINLOC (ARRAY [, MASK,KIND])

Description. Determine the location of the first element of ARRAY along dimension DIM having the minimum value of the elements identified by MASK.

Class. Transformational function.

Arguments.

ARRAY shall be of type integer, real, or character. It shall not be scalar.

DIM	shall be scalar and of type integer with a value in the range $1 < \text{DIM} \le n$, where n is the rank of ARRAY. The corresponding actual argument shall not be an optional dummy argument.
MASK (optional)	shall be of type logical and shall be conformable with ARRAY.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type. If DIM is absent, the result is an array of rank one and of size equal to the rank of ARRAY; otherwise, the result is of rank $n-1$ and shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$, where $(d_1, d_2, ..., d_n)$ is the shape of ARRAY.

Result Value.

Case (i): The result of MINLOC (ARRAY) is a rank-one array whose element values are the values of the subscripts of an element of ARRAY whose value equals the minimum value of all the elements of ARRAY. The *i*th subscript returned lies in the range 1 to e_i, where e_i is the extent of the *i*th dimension of ARRAY. If more than one element has the minimum value, the element whose subscripts are returned is the first such element, taken in array element order. If ARRAY has size zero, the value of the result is processor dependent.

Case (ii): The result of MINLOC (ARRAY, MASK = MASK) is a rank-one array whose element values are the values of the subscripts of an element of ARRAY, corresponding to a true element of MASK, whose value equals the minimum value of all such elements of ARRAY. The *i*th subscript returned lies in the range 1 to e_i, where e_i is the extent of the *i*th dimension of ARRAY. If more than one such element has the minimum value, the element whose subscripts are returned is the first such element taken in array element order. If ARRAY has size zero or every element of MASK has the value false, the value of the result is processor dependent.

Case (iii): If ARRAY has rank one, MINLOC (ARRAY, DIM = DIM [, MASK = MASK]) is a scalar whose value is equal to that of the first element of MINLOC (ARRAY [, MASK = MASK]). Otherwise, the value of element $(s_1, s_2, ..., s_{\text{DIM}-1}, s_{\text{DIM}+1}, ..., s_n)$ of the result is equal to

MINLOC (ARRAY $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$, DIM=1
[, MASK = MASK $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$]).

If ARRAY has type character, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the arguments is applied.

Examples.

Case (i): The value of MINLOC ((/ 4, 3, 6, 3 /)) is [2].

Case (ii): If A has the value $\begin{bmatrix} 0 & -5 & 8 & -3 \\ 3 & 4 & -1 & 2 \\ 1 & 5 & 6 & -4 \end{bmatrix}$, MINLOC (A, MASK = A **.**GT**.** –4) has the value [1, 4]. Note that this is true even if A has a declared lower bound other than 1.

Case (iii): The value of MINLOC ((/ 5, -9, 3 /), DIM = 1) is 2. If B has the value $\begin{bmatrix} 1 & 3 & -9 \\ 2 & 2 & 6 \end{bmatrix}$, MINLOC (B, DIM = 1) is [1, 2, 1] and MINLOC (B, DIM = 2) is [3, 1]. Note that this is true even if B has a declared lower bound other than 1.

13.11.77 MINVAL (ARRAY, DIM [, MASK]) or MINVAL (ARRAY [, MASK])

Description. Minimum value of all the elements of ARRAY along dimension DIM corresponding to true elements of MASK.

Class. Transformational function.

Arguments.

ARRAY	shall be of type integer, real, or character. It shall not be scalar.
DIM	shall be scalar and of type integer with a value in the range $1 \le \text{DIM} \le n$, where n is the rank of ARRAY. The corresponding actual argument shall not be an optional dummy argument.
MASK (optional)	shall be of type logical and shall be conformable with ARRAY.

Result Characteristics. The result is of the same type and type parameters as ARRAY. It is scalar if DIM is absent or ARRAY has rank one; otherwise, the result is an array of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of ARRAY.

Result Value.

Case (i): The result of MINVAL (ARRAY) has a value equal to the minimum value of all the elements of ARRAY if the size of ARRAY is not zero. If ARRAY has size zero and type integer or real, the result has the value of the positive number of the largest magnitude supported by the processor for numbers of the type and kind type parameter of ARRAY. If ARRAY has size zero and type character, the result has the value of a string of characters of length LEN (ARRAY), with each character equal to CHAR ($n-1$, KIND = KIND (ARRAY)), where n is the number of characters in the collating sequence for characters with the kind type parameter of ARRAY.

Case (ii): The result of MINVAL (ARRAY, MASK = MASK) has a value equal to that of MINVAL (PACK (ARRAY, MASK)).

Case (iii): The result of MINVAL (ARRAY, DIM = DIM [, MASK = MASK]) has a value equal to that of MINVAL (ARRAY [, MASK = MASK]) if ARRAY has rank one. Otherwise, the value of element $(s_1, s_2, ..., s_{\text{DIM}-1}, s_{\text{DIM}+1}, ..., s_n)$ of the result is equal to

MINVAL (ARRAY $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$
[, MASK= MASK $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$]).

If ARRAY has type character, the result is the value that would be selected by application of intrinsic relational operators; that is, the collating sequence for characters with the kind type parameter of the arguments is applied.

Examples.

Case (i): The value of MINVAL ((/ 1, 2, 3 /)) is 1.

Case (ii): MINVAL (C, MASK = C **.**GT**.** 0.0) forms the minimum of the positive elements of C.

Case (iii): If B is the array $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, MINVAL (B, DIM = 1) is [1, 3, 5] and MINVAL (B, DIM = 2) is [1, 2].

13.11.78 MOD (A, P)

Description. Remainder function.

Class. Elemental function.

Arguments.

A	shall be of type integer or real.
P	shall be of the same type and kind type parameter as A.

Result Characteristics. Same as A.

Result Value. If $P \neq 0$, the value of the result is A–INT (A / P) * P. If P = 0, the result is processor dependent.

Examples. MOD (3.0, 2.0) has the value 1.0 (approximately). MOD (8, 5) has the value 3. MOD (–8, 5) has the value –3. MOD (8, –5) has the value 3. MOD (–8, –5) has the value –3.

13.11.79 MODULO (A, P)

Description. Modulo function.

Class. Elemental function.

Arguments.

A	shall be of type integer or real.
P	shall be of the same type and kind type parameter as A.

Result Characteristics. Same as A.

Result Value.

Case (i): A is of type integer. If $P \neq 0$, MODULO (A, P) has the value R such that $A = Q \times P + R$, where Q is an integer, the inequalities $0 \leq R < P$ hold if $P > 0$, and $P < R \leq 0$ hold if $P < 0$. If P = 0, the result is processor dependent.

Case (ii): A is of type real. If $P \neq 0$, the value of the result is A – FLOOR (A / P) * P. If P = 0, the result is processor dependent.

Examples. MODULO (8, 5) has the value 3. MODULO (–8, 5) has the value 2. MODULO (8, –5) has the value –2. MODULO (–8, –5) has the value –3.

13.11.80 MVBITS (FROM, FROMPOS, LEN, TO, TOPOS)

Description. Copies a sequence of bits from one data object to another.

Class. Elemental subroutine.

Arguments.

FROM	shall be of type integer. It is an INTENT (IN) argument.
FROMPOS	shall be of type integer and nonnegative. It is an INTENT (IN) argument. FROMPOS + LEN shall be less than or equal to BIT_SIZE (FROM). The model for the interpretation of an integer value as a sequence of bits is in 13.4.
LEN	shall be of type integer and nonnegative. It is an INTENT (IN) argument.
TO	shall be a variable of type integer with the same kind type parameter value as FROM and may be the same variable as FROM. It is an INTENT (INOUT) argument. TO is defined by copying the sequence of bits of length LEN, starting at position FROMPOS of FROM to position TOPOS of TO. No other bits of TO are altered. On return, the LEN bits of TO starting at TOPOS are equal to the value that the LEN bits of FROM starting at FROMPOS had on entry. The model for the interpretation of an integer value as a sequence of bits is in 13.4.
TOPOS	shall be of type integer and nonnegative. It is an INTENT (IN) argument. TOPOS + LEN shall be less than or equal to BIT_SIZE (TO).

Example. If TO has the initial value 6, the value of TO after the statement CALL MVBITS (7, 2, 2, TO, 0) is 5.

13.11.81 NEAREST (X, S)

Description. Returns the nearest different machine representable number in a given direction.

Class. Elemental function.

Arguments.

X	shall be of type real.
S	shall be of type real and not equal to zero.

Result Characteristics. Same as X.

Result Value. The result has a value equal to the machine representable number distinct from X and nearest to it in the direction of the infinity with the same sign as S.

Example. NEAREST (3.0, 2.0) has the value $3 + 2^{-22}$ on a machine whose representation is that of the model at the end of 13.5.

NOTE 13.17

Unlike other floating-point manipulation functions, NEAREST operates on machine representatable numbers rather than model numbers. On many systems there are machine-representable numbers that lie between adjacent model numbers.

13.11.82 NINT (A [, KIND])

Description. Nearest integer.

Class. Elemental function.

Arguments.

A	shall be of type real.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise, the kind type parameter is that of default integer type.

Result Value. If A > 0, NINT (A) has the value INT (A+0.5); if A ≤ 0, NINT (A) has the value INT (A–0.5).

Example. NINT (2.783) has the value 3.

13.11.83 NOT (I)

Description. Performs a bitwise complement.

Class. Elemental function.

Argument. I shall be of type integer.

Result Characteristics. Same as I.

Result Value. The result has the value obtained by complementing I bit-by-bit according to the following truth table:

I	NOT (I)
1	0
0	1

The model for the interpretation of an integer value as a sequence of bits is in 13.4.

Example. If I is represented by the string of bits 01010101, NOT (I) has the binary value 10101010.

13.11.84 NULL ([MOLD])

Description. Returns a disassociated pointer, designates an unallocated allocatable component of a structure constructor, or designates a deferred type-bound procedure binding.

Class. Transformational function.

Argument. MOLD shall be a pointer. It may be of any type or may be a procedure pointer. Its pointer association status may be undefined, disassociated, or associated. If its status is associated, the target need not be defined with a value.

Result Characteristics. If MOLD is present, the characteristics are the same as MOLD. If MOLD has deferred type parameters, those type parameters of the result are deferred.

If MOLD is absent, the characteristics of the result are determined by the entity with which the reference is associated. See Table 13.1. MOLD shall not be absent in any other context. If any type parameters of the contextual entity are deferred, those type parameters of the result are deferred. If any type parameters of the contextual entity are assumed, MOLD shall be present.

If the context of the reference to NULL is an actual argument to a generic procedure, MOLD shall be present if the type, type parameters, or rank is required to resolve the generic reference.

Table 13.1 Characteristics of the result of NULL ( )

Appearance of NULL ( )	Type, type parameters, and rank of result:
right side of a pointer assignment	pointer on the left side
initialization for an object in a declaration	the object
default initialization for a component	the component
in a structure constructor	the corresponding component
as an actual argument	the corresponding dummy argument
in a DATA statement	the corresponding pointer object
in a *proc-binding-stmt* for which no *proc-interface-name* is specified	the interface of the type-bound procedure that would have been inherited
in a *proc-binding-stmt* for which a *proc-interface-name* is specified	the interface specified by the *proc-binding-stmt*.

Result. The result is a disassociated pointer, an unallocated allocatable entity, or a deferred type-bound procedure binding.

Example. REAL, POINTER, DIMENSION(:) :: VEC => NULL ( ) defines the initial association status of VEC to be disassociated. The MOLD argument is required in the following:

```
INTERFACE GEN
    SUBROUTINE S1 (J, PI)
        INTEGER J
        INTEGER, POINTER :: PI
    END SUBROUTINE S1
    SUBROUTINE S2 (K, PR)
        INTEGER K
        REAL, POINTER :: PR
    END SUBROUTINE S2
END INTERFACE
```

```
REAL, POINTER :: REAL_PTR
CALL GEN (7, NULL (REAL_PTR) )      ! Invokes S2
```

13.11.85 PACK (ARRAY, MASK [, VECTOR])

Description. Pack an array into an array of rank one under the control of a mask.

Class. Transformational function.

Arguments.

ARRAY	may be of any type. It shall not be scalar.
MASK	shall be of type logical and shall be conformable with ARRAY.
VECTOR (optional)	shall be of the same type and type parameters as ARRAY and shall have rank one. VECTOR shall have at least as many elements as there are true elements in MASK. If MASK is scalar with the value true, VECTOR shall have at least as many elements as there are in ARRAY.

Result Characteristics. The result is an array of rank one with the same type and type parameters as ARRAY. If VECTOR is present, the result size is that of VECTOR; otherwise, the result size is the number t of true elements in MASK unless MASK is scalar with the value true, in which case the result size is the size of ARRAY.

Result Value. Element i of the result is the element of ARRAY that corresponds to the ith true element of MASK, taking elements in array element order, for $i = 1, 2, ..., t$. If VECTOR is present and has size $n > t$, element i of the result has the value VECTOR (i), for $i = t + 1, ..., n$.

Examples. The nonzero elements of an array M with the value $\begin{bmatrix} 0 & 0 & 0 \\ 9 & 0 & 0 \\ 0 & 0 & 7 \end{bmatrix}$ may be "gathered" by the function PACK. The result of PACK (M, MASK = M **.**NE**.** 0) is [9, 7] and the result of PACK (M, M **.**NE**.** 0, VECTOR = (/ 2, 4, 6, 8, 10, 12 /)) is [9, 7, 6, 8, 10, 12].

13.11.86 PRECISION (X)

Description. Returns the decimal precision of the model representing real numbers with the same kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type real or complex. It may be scalar or array valued.

Result Characteristics. Default integer scalar.

Result Value. The result has the value INT (($p - 1$) $*$ LOG10 (b)) + k, where b and p are as defined in 13.5 for the model representing real numbers with the same value for the kind type parameter as X, and where k is 1 if b is an integral power of 10 and 0 otherwise.

Example. PRECISION (X) has the value INT (23 $*$ LOG10 (2.)) = INT (6.92...) = 6 for real X whose model is as at the end of 13.5.

13.11.87 PRESENT (A)

Description. Determine whether an optional argument is present.

Class. Inquiry function.

Argument. A shall be the name of an optional dummy argument that is accessible in the subprogram in which the PRESENT function reference appears. It may be of any type and it

may be a pointer. It may be scalar or array valued. It may be a dummy procedure. The dummy argument A has no INTENT attribute.

Result Characteristics. Default logical scalar.

Result Value. The result has the value true if A is present (12.4.1.6) and otherwise has the value false.

13.11.88 PRODUCT (ARRAY, DIM [, MASK]) or PRODUCT (ARRAY [, MASK])

Description. Product of all the elements of ARRAY along dimension DIM corresponding to the true elements of MASK.

Class. Transformational function.

Arguments.

ARRAY	shall be of type integer, real, or complex. It shall not be scalar.
DIM	shall be scalar and of type integer with a value in the range $1 \le \text{DIM} \le n$, where n is the rank of ARRAY. The corresponding actual argument shall not be an optional dummy argument.
MASK (optional)	shall be of type logical and shall be conformable with ARRAY.

Result Characteristics. The result is of the same type and kind type parameter as ARRAY. It is scalar if DIM is absent or ARRAY has rank one; otherwise, the result is an array of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of ARRAY.

Result Value.

Case (i): The result of PRODUCT (ARRAY) has a value equal to a processor-dependent approximation to the product of all the elements of ARRAY or has the value one if ARRAY has size zero.

Case (ii): The result of PRODUCT (ARRAY, MASK = MASK) has a value equal to a processor-dependent approximation to the product of the elements of ARRAY corresponding to the true elements of MASK or has the value one if there are no true elements.

Case (iii): If ARRAY has rank one, PRODUCT (ARRAY, DIM = DIM [, MASK = MASK]) has a value equal to that of PRODUCT (ARRAY [, MASK = MASK]). Otherwise, the value of element $(s_1, s_2, ..., s_{\text{DIM}-1}, s_{\text{DIM}+1}, ..., s_n)$ of PRODUCT (ARRAY, DIM = DIM [,MASK = MASK]) is equal to

PRODUCT (ARRAY $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$
[, MASK = MASK $(s_1, s_2, ..., s_{\text{DIM}-1}, :, s_{\text{DIM}+1}, ..., s_n)$]).

Examples.

Case (i): The value of PRODUCT ((/ 1, 2, 3 /)) is 6.

Case (ii): PRODUCT (C, MASK = C **.**GT**.** 0.0) forms the product of the positive elements of C.

Case (iii): If B is the array $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, PRODUCT (B, DIM = 1) is [2, 12, 30] and PRODUCT (B, DIM = 2) is [15, 48].

13.11.89 RADIX (X)

Description. Returns the base of the model representing numbers of the same type and kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type integer or real. It may be scalar or array valued.

Result Characteristics. Default integer scalar.

Result Value. The result has the value r if X is of type integer and the value b if X is of type real, where r and b are as defined in 13.5 for the model representing numbers of the same type and kind type parameter as X.

Example. RADIX (X) has the value 2 for real X whose model is as at the end of 13.5.

13.11.90 RANDOM_NUMBER (HARVEST)

Description. Returns one pseudorandom number or an array of pseudorandom numbers from the uniform distribution over the range $0 \le x < 1$.

Class. Subroutine.

Argument. HARVEST shall be of type real. It is an INTENT (OUT) argument. It may be a scalar or an array variable. It is assigned pseudorandom numbers from the uniform distribution in the interval $0 \le x < 1$.

Examples.

```
REAL X, Y (10, 10)
! Initialize X with a pseudorandom number
CALL RANDOM_NUMBER (HARVEST = X)
CALL RANDOM_NUMBER (Y)
! X and Y contain uniformly distributed random numbers
```

13.11.91 RANDOM_SEED ([SIZE, PUT, GET])

Description. Restarts or queries the pseudorandom number generator used by RANDOM_NUMBER.

Class. Subroutine.

Arguments. There shall either be exactly one or no arguments present.

SIZE (optional) shall be scalar and of type default integer. It is an INTENT (OUT) argument. It is assigned the number N of integers that the processor uses to hold the value of the seed.

PUT (optional) shall be a default integer array of rank one and size $\ge N$. It is an INTENT (IN) argument. It is used in a processor-dependent manner to compute the seed value accessed by the pseudorandom number generator.

GET (optional) shall be a default integer array of rank one and size $\ge N$ It is an INTENT (OUT) argument. It is assigned the current value of the seed.

If no argument is present, the processor assigns a processor-dependent value to the seed.

The pseudorandom number generator used by RANDOM_NUMBER maintains a seed that is updated during the execution of RANDOM_NUMBER and that may be specified or returned by RANDOM_SEED. Computation of the seed from the argument PUT is performed in a processor-dependent manner. The value returned by GET need not be the same as the value specified by PUT in an immediately preceding reference to RANDOM_SEED. For example, following execution of the statements

```
CALL RANDOM_SEED (PUT=SEED1)
CALL RANDOM_SEED (GET=SEED2)
```

SEED2 need not equal SEED1. When the values differ, the use of either value as the PUT argument in a subsequent call to RANDOM_SEED shall result in the same sequence of pseudorandom numbers being generated. For example, after execution of the statements

```
CALL RANDOM_SEED (PUT=SEED1)
CALL RANDOM_SEED (GET=SEED2)
CALL RANDOM_NUMBER (X1)
CALL RANDOM_SEED (PUT=SEED2)
CALL RANDOM_NUMBER (X2)
```

X2 equals X1.

Examples.

```
CALL RANDOM_SEED                      ! Processor initialization
CALL RANDOM_SEED (SIZE = K)           ! Puts size of seed in K
CALL RANDOM_SEED (PUT = SEED (1 : K)) ! Define seed
CALL RANDOM_SEED (GET = OLD (1 : K))  ! Read current seed
```

13.11.92 RANGE (X)

Description. Returns the decimal exponent range of the model representing integer or real numbers with the same kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type integer, real, or complex. It may be scalar or array valued.

Result Characteristics. Default integer scalar.

Result Value.

Case (i): For an integer argument, the result has the value INT (LOG10 (HUGE(X))).

Case (ii): For a real or complex argument, the result has the value INT (MIN (LOG10 (HUGE(X)), –LOG10 (TINY(X)))).

Examples. RANGE (X) has the value 38 for real X whose model is as at the end of 13.5, since in this case HUGE(X) = $(1 - 2^{-24}) \times 2^{127}$ and TINY(X) = 2^{-127}.

13.11.93 REAL (A [, KIND])

Description. Convert to real type.

Class. Elemental function.

Arguments.

A	shall be of type integer, real, or complex.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Real.

Case (i): If A is of type integer or real and KIND is present, the kind type parameter is that specified by the value of KIND. If A is of type integer or real and KIND is not present, the kind type parameter is the processor-dependent kind type parameter for the default real type.

Case (ii): If A is of type complex and KIND is present, the kind type parameter is that specified by the value of KIND. If A is of type complex and KIND is not present, the kind type parameter is the kind type parameter of A.

Result Value.

Case (i): If A is of type integer or real, the result is equal to a processor-dependent approximation to A.

Case (ii): If A is of type complex, the result is equal to a processor-dependent approximation to the real part of A.

Examples. REAL (–3) has the value –3.0. REAL (Z) has the same kind type parameter and the same value as the real part of the complex variable Z.

13.11.94 REPEAT (STRING, NCOPIES)

Description. Concatenate several copies of a string.

Class. Transformational function.

Arguments.

STRING	shall be scalar and of type character.
NCOPIES	shall be scalar and of type integer. Its value shall not be negative.

Result Characteristics. Character scalar of length NCOPIES times that of STRING, with the same kind type parameter as STRING.

Result Value. The value of the result is the concatenation of NCOPIES copies of STRING.

Examples. REPEAT ('H', 2) has the value HH. REPEAT ('XYZ', 0) has the value of a zero-length string.

13.11.95 RESHAPE (SOURCE, SHAPE [, PAD, ORDER])

Description. Constructs an array of a specified shape from the elements of a given array.

Class. Transformational function.

Arguments.

SOURCE	may be of any type. It shall not be scalar. If PAD is absent or of size zero, the size of SOURCE shall be greater than or equal to PRODUCT (SHAPE). The size of the result is the product of the values of the elements of SHAPE.
SHAPE	shall be of type integer, rank one, and constant size. Its size shall be positive and less than 8. It shall not have an element whose value is negative.
PAD (optional)	shall be of the same type and type parameters as SOURCE. PAD shall not be scalar.
ORDER (optional)	shall be of type integer, shall have the same shape as SHAPE, and its value shall be a permutation of (1, 2, ..., *n*), where *n* is the size of SHAPE. If absent, it is as if it were present with value (1, 2, ..., *n*).

Result Characteristics. The result is an array of shape SHAPE (that is, SHAPE (RESHAPE (SOURCE, SHAPE, PAD, ORDER)) is equal to SHAPE) with the same type and type parameters as SOURCE.

Result Value. The elements of the result, taken in permuted subscript order ORDER (1), ..., ORDER (*n*), are those of SOURCE in normal array element order followed if necessary by those of PAD in array element order, followed if necessary by additional copies of PAD in array element order.

Examples. RESHAPE ((/ 1, 2, 3, 4, 5, 6 /), (/ 2, 3 /)) has the value $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$.

RESHAPE ((/ 1, 2, 3, 4, 5, 6 /), (/ 2, 4 /), (/ 0, 0 /), (/ 2, 1 /)) has the value $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 0 & 0 \end{bmatrix}$.

13.11.96 RRSPACING (X)

Description. Returns the reciprocal of the relative spacing of model numbers near the argument value.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Same as X.

Result Value. The result has the value $|X \times b^{-e}| \times b^{p}$, where b, e, and p are as defined in 13.5 for the model representation of X.

Example. RRSPACING (–3.0) has the value 0.75×2^{24} for reals whose model is as at the end of 13.5.

13.11.97 SAME_TYPE_AS (A, B)

Description. Inquires whether the dynamic type of A is the same as the dynamic type of B.

Class. Inquiry function.

Arguments.

A	shall be an object of extensible type. If it is a pointer, it shall not have an undefined association status.
B	shall be an object of extensible type. If it is a pointer, it shall not have an undefined association status.

Result Characteristics. Default logical scalar.

Result Value. If either A or B is unlimited polymorphic and is a disassociated pointer or an unallocated allocatable, the result is true if and only if both are; otherwise the result is true if and only if the dynamic type of A is the same as the dynamic type of B.

NOTE 13.18

If either A or B is a disassociated pointer, the dynamic type is the same as the declared type.

13.11.98 SCALE (X, I)

Description. Returns $X \times b^{I}$ where b is the base of the model representation of X.

Class. Elemental function.

Arguments.

X	shall be of type real.
I	shall be of type integer.

Result Characteristics. Same as X.

Result Value. The result has the value $X \times b^{I}$, where b is defined in 13.5 for model numbers representing values of X, provided this result is within range; if not, the result is processor dependent.

Example. SCALE (3.0, 2) has the value 12.0 for reals whose model is as at the end of 13.5.

13.11.99 SCAN (STRING, SET [, BACK, KIND])

Description. Scan a string for any one of the characters in a set of characters.

Class. Elemental function.

Arguments.

STRING	shall be of type character.

SET	shall be of type character with the same kind type parameter as STRING.
BACK (optional)	shall be of type logical.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type.

Result Value.

Case (i): If BACK is absent or is present with the value false and if STRING contains at least one character that is in SET, the value of the result is the position of the leftmost character of STRING that is in SET.

Case (ii): If BACK is present with the value true and if STRING contains at least one character that is in SET, the value of the result is the position of the rightmost character of STRING that is in SET.

Case (iii): The value of the result is zero if no character of STRING is in SET or if the length of STRING or SET is zero.

Examples.

Case (i): SCAN ('FORTRAN', 'TR') has the value 3.

Case (ii): SCAN ('FORTRAN', 'TR', BACK = **.**TRUE**.**) has the value 5.

Case (iii): SCAN ('FORTRAN', 'BCD') has the value 0.

13.11.100 SELECTED_CHAR_KIND (NAME)

Description. Returns the value of the kind type parameter of the character set named by the argument.

Class. Transformational function.

Argument. NAME shall be scalar and of type default character.

Result Characteristics. Default integer scalar.

Result Value. If NAME has the value DEFAULT, then the result has a value equal to that of the kind type parameter of the default character data type. If NAME has the value ASCII, then the result has a value equal to that of the kind type parameter of the ASCII character data type if the processor supports such a type; otherwise the result has the value -1. If NAME has the value ISO_10646, then the result has a value equal to that of the kind type parameter of the ISO/IEC 10646-1:2000 UCS-4 character data type if the processor supports such a type; otherwise the result has the value -1. If NAME is a processor-defined name of some other character type supported by the processor, then the result has a value equal to that of the kind type parameter of that character type. If NAME is not the name of a supported character type, then the result has the value -1. The NAME is interpreted without respect to case or trailing blanks.

Example. SELECTED_CHAR_KIND ('ASCII') has the value 1 on a processor that uses 1 as the kind type parameter for the ASCII character set.

NOTE 13.19

ISO_10646 refers to the UCS-4 representation, a 4-octet character set.

13.11.101 SELECTED_INT_KIND (R)

Description. Returns a value of the kind type parameter of an integer data type that represents all integer values n with $-10^R < n < 10^R$.

Class. Transformational function.

Argument. R shall be scalar and of type integer.

Result Characteristics. Default integer scalar.

Result Value. The result has a value equal to the value of the kind type parameter of an integer data type that represents all values n in the range $-10^R < n < 10^R$, or if no such kind type parameter is available on the processor, the result is –1. If more than one kind type parameter meets the criteria, the value returned is the one with the smallest decimal exponent range, unless there are several such values, in which case the smallest of these kind values is returned.

Example. SELECTED_INT_KIND (6) has the value KIND (0) on a machine that supports a default integer representation method with $r = 2$ and $q = 31$.

13.11.102 SELECTED_REAL_KIND ([P, R])

Description. Returns a value of the kind type parameter of a real data type with decimal precision of at least P digits and a decimal exponent range of at least R.

Class. Transformational function.

Arguments. At least one argument shall be present.

P (optional) shall be scalar and of type integer.

R (optional) shall be scalar and of type integer.

Result Characteristics. Default integer scalar.

Result Value. The result has a value equal to a value of the kind type parameter of a real data type with decimal precision, as returned by the function PRECISION, of at least P digits and a decimal exponent range, as returned by the function RANGE, of at least R, or if no such kind type parameter is available on the processor, the result is –1 if the processor does not support a real data type with a precision greater than or equal to P, –2 if the processor does not support a real type with an exponent range greater than or equal to R, and –3 if neither is supported. If more than one kind type parameter value meets the criteria, the value returned is the one with the smallest decimal precision, unless there are several such values, in which case the smallest of these kind values is returned.

Example. SELECTED_REAL_KIND (6, 70) has the value KIND (0.0) on a machine that supports a default real approximation method with $b = 16$, $p = 6$, $e_{\min} = -64$, and $e_{\max} = 63$.

13.11.103 SET_EXPONENT (X, I)

Description. Returns the model number whose fractional part is the fractional part of the model representation of X and whose exponent part is I.

Class. Elemental function.

Arguments.

X shall be of type real.

I shall be of type integer.

Result Characteristics. Same as X.

Result Value. The result has the value $X \times b^{I-e}$, where b and e are as defined in 13.5 for the model representation of X. If X has value zero, the result has value zero.

Example. SET_EXPONENT (3.0, 1) has the value 1.5 for reals whose model is as at the end of 13.5.

13.11.104 SHAPE (SOURCE [, KIND])

Description. Returns the shape of an array or a scalar.

Class. Inquiry function.

Arguments.

SOURCE	may be of any type. It may be array valued or scalar. It shall not be an unallocated allocatable or a pointer that is not associated. It shall not be an assumed-size array.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type. The result is an array of rank one whose size is equal to the rank of SOURCE.

Result Value. The value of the result is the shape of SOURCE.

Examples. The value of SHAPE (A (2:5, –1:1)) is [4, 3]. The value of SHAPE (3) is the rank-one array of size zero.

13.11.105 SIGN (A, B)

Description. Absolute value of A times the sign of B.

Class. Elemental function.

Arguments.

A	shall be of type integer or real.
B	shall be of the same type and kind type parameter as A.

Result Characteristics. Same as A.

Result Value.

Case (i): If B > 0, the value of the result is |A|.

Case (ii): If B < 0, the value of the result is -|A|.

Case (iii): If B is of type integer and B=0, the value of the result is |A|.

Case (iv): If B is of type real and is zero, then

(a) If the processor cannot distinguish between positive and negative real zero, the value of the result is |A|.

(b) If B is positive real zero, the value of the result is |A|.

(c) If B is negative real zero, the value of the result is -|A|.

Example. SIGN (–3.0, 2.0) has the value 3.0.

13.11.106 SIN (X)

Description. Sine function.

Class. Elemental function.

Argument. X shall be of type real or complex.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to sin(X). If X is of type real, it is regarded as a value in radians. If X is of type complex, its real part is regarded as a value in radians.

Example. SIN (1.0) has the value 0.84147098 (approximately).

13.11.107 SINH (X)

Description. Hyperbolic sine function.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to sinh(X).

Example. SINH (1.0) has the value 1.1752012 (approximately).

13.11.108 SIZE (ARRAY [, DIM, KIND])

Description. Returns the extent of an array along a specified dimension or the total number of elements in the array.

Class. Inquiry function.

Arguments.

ARRAY	may be of any type. It shall not be scalar. It shall not be an unallocated allocatable or a pointer that is not associated. If ARRAY is an assumed-size array, DIM shall be present with a value less than the rank of ARRAY.
DIM (optional)	shall be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where n is the rank of ARRAY.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer scalar. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type.

Result Value. The result has a value equal to the extent of dimension DIM of ARRAY or, if DIM is absent, the total number of elements of ARRAY.

Examples. The value of SIZE (A (2:5, –1:1), DIM=2) is 3. The value of SIZE (A (2:5, –1:1)) is 12.

13.11.109 SPACING (X)

Description. Returns the absolute spacing of model numbers near the argument value.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Same as X.

Result Value. If X is not zero, the result has the value b^{e-p}, where b, e, and p are as defined in 13.5 for the model representation of X, provided this result is within range. Otherwise, the result is the same as that of TINY (X).

Example. SPACING (3.0) has the value 2^{-22} for reals whose model is as at the end of 13.5.

13.11.110 SPREAD (SOURCE, DIM, NCOPIES)

Description. Replicates an array by adding a dimension. Broadcasts several copies of SOURCE along a specified dimension (as in forming a book from copies of a single page) and thus forms an array of rank one greater.

Class. Transformational function.

Arguments.

SOURCE	may be of any type. It may be scalar or array valued. The rank of SOURCE shall be less than 7.

DIM shall be scalar and of type integer with value in the range $1 \le \text{DIM} \le n+1$, where n is the rank of SOURCE.

NCOPIES shall be scalar and of type integer.

Result Characteristics. The result is an array of the same type and type parameters as SOURCE and of rank $n+1$, where n is the rank of SOURCE.

Case (i): If SOURCE is scalar, the shape of the result is (MAX (NCOPIES, 0)).

Case (ii): If SOURCE is array valued with shape $(d_1, d_2, ..., d_n)$, the shape of the result is $(d_1, d_2, ..., d_{\text{DIM}-1}, \text{MAX (NCOPIES, 0)}, d_{\text{DIM}}, ..., d_n)$.

Result Value.

Case (i): If SOURCE is scalar, each element of the result has a value equal to SOURCE.

Case (ii): If SOURCE is array valued, the element of the result with subscripts $(r_1, r_2, ..., r_{n+1})$ has the value SOURCE $(r_1, r_2, ..., r_{\text{DIM}-1}, r_{\text{DIM}+1}, ..., r_{n+1})$.

Examples. If A is the array [2, 3, 4], SPREAD (A, DIM=1, NCOPIES=NC) is the array $\begin{bmatrix} 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix}$ if NC has the value 3 and is a zero-sized array if NC has the value 0.

13.11.111 SQRT (X)

Description. Square root.

Class. Elemental function.

Argument. X shall be of type real or complex. Unless X is complex, its value shall be greater than or equal to zero.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to the square root of X. A result of type complex is the principal value with the real part greater than or equal to zero. When the real part of the result is zero, the imaginary part is greater than or equal to zero.

Example. SQRT (4.0) has the value 2.0 (approximately).

13.11.112 SUM (ARRAY, DIM [, MASK]) or SUM (ARRAY [, MASK])

Description. Sum all the elements of ARRAY along dimension DIM corresponding to the true elements of MASK.

Class. Transformational function.

Arguments.

ARRAY shall be of type integer, real, or complex. It shall not be scalar.

DIM shall be scalar and of type integer with a value in the range $1 \le \text{DIM} \le n$, where n is the rank of ARRAY. The corresponding actual argument shall not be an optional dummy argument.

MASK (optional) shall be of type logical and shall be conformable with ARRAY.

Result Characteristics. The result is of the same type and kind type parameter as ARRAY. It is scalar if DIM is absent or ARRAY has rank one; otherwise, the result is an array of rank $n-1$ and of shape $(d_1, d_2, ..., d_{\text{DIM}-1}, d_{\text{DIM}+1}, ..., d_n)$ where $(d_1, d_2, ..., d_n)$ is the shape of ARRAY.

Result Value.

Case (i): The result of SUM (ARRAY) has a value equal to a processor-dependent approximation to the sum of all the elements of ARRAY or has the value zero if ARRAY has size zero.

Case (ii): The result of SUM (ARRAY, MASK = MASK) has a value equal to a processor-dependent approximation to the sum of the elements of ARRAY corresponding to the true elements of MASK or has the value zero if there are no true elements.

Case (iii): If ARRAY has rank one, SUM (ARRAY, DIM = DIM [, MASK = MASK]) has a value equal to that of SUM (ARRAY [,MASK = MASK]). Otherwise, the value of element $(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_n)$ of
SUM (ARRAY, DIM = DIM [, MASK = MASK]) is equal to

SUM (ARRAY $(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_n)$
[, MASK= MASK $(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_n)$]).

Examples.

Case (i): The value of SUM ((/ 1, 2, 3 /)) is 6.

Case (ii): SUM (C, MASK= C **.**GT**.** 0.0) forms the arithmetic sum of the positive elements of C.

Case (iii): If B is the array $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, SUM (B, DIM = 1) is [3, 7, 11] and SUM (B, DIM = 2) is [9, 12].

13.11.113 SYSTEM_CLOCK ([COUNT, COUNT_RATE, COUNT_MAX])

Description. Returns numeric data from a real-time clock.

Class. Subroutine.

Arguments.

COUNT (optional) shall be scalar and of type integer. It is an INTENT (OUT) argument. It is assigned a processor-dependent value based on the current value of the processor clock, or –HUGE (COUNT) if there is no clock. The processor-dependent value is incremented by one for each clock count until the value COUNT_MAX is reached and is reset to zero at the next count. It lies in the range 0 to COUNT_MAX if there is a clock.

COUNT_RATE (optional)
shall be scalar and of type integer or real. It is an INTENT (OUT) argument. It is assigned a processor-dependent approximation to the number of processor clock counts per second, or zero if there is no clock.

COUNT_MAX (optional)
shall be scalar and of type integer. It is an INTENT (OUT) argument. It is assigned the maximum value that COUNT can have, or zero if there is no clock.

Example. If the processor clock is a 24-hour clock that registers time at approximately 18.20648193 ticks per second, at 11:30 A.M. the reference

```
CALL SYSTEM_CLOCK (COUNT = C, COUNT_RATE = R, COUNT_MAX = M)
```

defines C = $(11 \times 3600 + 30 \times 60) \times 18.20648193 = 753748$, R = 18.20648193, and
M = $24 \times 3600 \times 18.20648193 - 1 = 86399$.

13.11.114 TAN (X)

Description. Tangent function.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to tan(X), with X regarded as a value in radians.

Example. TAN (1.0) has the value 1.5574077 (approximately).

13.11.115 TANH (X)

Description. Hyperbolic tangent function.

Class. Elemental function.

Argument. X shall be of type real.

Result Characteristics. Same as X.

Result Value. The result has a value equal to a processor-dependent approximation to tanh(X).

Example. TANH (1.0) has the value 0.76159416 (approximately).

13.11.116 TINY (X)

Description. Returns the smallest positive number of the model representing numbers of the same type and kind type parameter as the argument.

Class. Inquiry function.

Argument. X shall be of type real. It may be scalar or array valued.

Result Characteristics. Scalar with the same type and kind type parameter as X.

Result Value. The result has the value $b^{e_{\min}-1}$ where b and $e_{\min}$ are as defined in 13.5 for the model representing numbers of the same type and kind type parameter as X.

Example. TINY (X) has the value 2^{-127} for real X whose model is as at the end of 13.5.

13.11.117 TRANSFER (SOURCE, MOLD [, SIZE])

Description. Returns a result with a physical representation identical to that of SOURCE but interpreted with the type and type parameters of MOLD.

Class. Transformational function.

Arguments.

SOURCE	may be of any type and may be scalar or array valued.
MOLD	may be of any type and may be scalar or array valued.
SIZE (optional)	shall be scalar and of type integer. The corresponding actual argument shall not be an optional dummy argument.

Result Characteristics. The result is of the same type and type parameters as MOLD.

Case (i): If MOLD is a scalar and SIZE is absent, the result is a scalar.

Case (ii): If MOLD is array valued and SIZE is absent, the result is array valued and of rank one. Its size is as small as possible such that its physical representation is not shorter than that of SOURCE.

Case (iii): If SIZE is present, the result is array valued of rank one and size SIZE.

Result Value. If the physical representation of the result has the same length as that of SOURCE, the physical representation of the result is that of SOURCE. If the physical representation of the result is longer than that of SOURCE, the physical representation of the leading part is that of SOURCE and the remainder is undefined. If the physical representation of the result is shorter than that of SOURCE, the physical representation of the result is the leading part of SOURCE. If D and E are scalar variables such that the physical representation of D is as long as or longer than that of E, the value of TRANSFER (TRANSFER (E, D), E) shall be the value of E. IF D is an array and E is an array of rank one, the value of TRANSFER (TRANSFER (E, D), E, SIZE (E)) shall be the value of E.

Examples.

Case (i): TRANSFER (1082130432, 0.0) has the value 4.0 on a processor that represents the values 4.0 and 1082130432 as the string of binary digits 0100 0000 1000 0000 0000 0000 0000 0000.

Case (ii): TRANSFER ((/ 1.1, 2.2, 3.3 /), (/ (0.0, 0.0) /)) is a complex rank-one array of length two whose first element has the value (1.1, 2.2) and whose second element has a real part with the value 3.3. The imaginary part of the second element is undefined.

Case (iii): TRANSFER ((/ 1.1, 2.2, 3.3 /), (/ (0.0, 0.0) /), 1) is a complex rank-one array of length one whose only element has the value (1.1, 2.2).

13.11.118 TRANSPOSE (MATRIX)

Description. Transpose an array of rank two.

Class. Transformational function.

Argument. MATRIX may be of any type and shall have rank two.

Result Characteristics. The result is an array of the same type and type parameters as MATRIX and with rank two and shape (n, m) where (m, n) is the shape of MATRIX.

Result Value. Element (i, j) of the result has the value MATRIX (j, i), $i = 1, 2, ..., n$; $j = 1, 2, ..., m$.

Example. If A is the array $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, then TRANSPOSE (A) has the value $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$.

13.11.119 TRIM (STRING)

Description. Returns the argument with trailing blank characters removed.

Class. Transformational function.

Argument. STRING shall be of type character and shall be a scalar.

Result Characteristics. Character with the same kind type parameter value as STRING and with a length that is the length of STRING less the number of trailing blanks in STRING.

Result Value. The value of the result is the same as STRING except any trailing blanks are removed. If STRING contains no nonblank characters, the result has zero length.

Example. TRIM (' A B ') has the value ' A B'.

13.11.120 UBOUND (ARRAY [, DIM, KIND])

Description. Returns all the upper bounds of an array or a specified upper bound.

Class. Inquiry function.

Arguments.

ARRAY may be of any type. It shall not be scalar. It shall not be an unallocated allocatable or a pointer that is not associated. If ARRAY is an assumed-size array, DIM shall be present with a value less than the rank of ARRAY.

DIM (optional) shall be scalar and of type integer with a value in the range $1 \leq \text{DIM} \leq n$, where n is the rank of ARRAY. The corresponding actual argument shall not be an optional dummy argument.

KIND (optional) shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type. It is scalar if DIM is present; otherwise, the result is an array of rank one and size n, where n is the rank of ARRAY.

Result Value.

Case (i): For an array section or for an array expression, other than a whole array or array structure component, UBOUND (ARRAY, DIM) has a value equal to the number of elements in the given dimension; otherwise, it has a value equal to the upper bound for subscript DIM of ARRAY if dimension DIM of ARRAY does not have size zero and has the value zero if dimension DIM has size zero.

Case (ii): UBOUND (ARRAY) has a value whose ith element is equal to UBOUND (ARRAY, i), for $i = 1, 2, ..., n$, where n is the rank of ARRAY.

Examples. If A is declared by the statement

```
REAL A (2:3, 7:10)
```

then UBOUND (A) is [3, 10] and UBOUND (A, DIM = 2) is 10.

13.11.121 UNPACK (VECTOR, MASK, FIELD)

Description. Unpack an array of rank one into an array under the control of a mask.

Class. Transformational function.

Arguments.

VECTOR may be of any type. It shall have rank one. Its size shall be at least t where t is the number of true elements in MASK.

MASK shall be array valued and of type logical.

FIELD shall be of the same type and type parameters as VECTOR and shall be conformable with MASK.

Result Characteristics. The result is an array of the same type and type parameters as VECTOR and the same shape as MASK.

Result Value. The element of the result that corresponds to the ith true element of MASK, in array element order, has the value VECTOR (i) for $i = 1, 2, ..., t$, where t is the number of true values in MASK. Each other element has a value equal to FIELD if FIELD is scalar or to the corresponding element of FIELD if it is an array.

Examples. Specific values may be "scattered" to specific positions in an array by using UNPACK. If M is the array $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, V is the array [1, 2, 3], and Q is the logical mask

$\begin{bmatrix} . & T & . \\ T & . & . \\ . & . & T \end{bmatrix}$, where "T" represents true and "." represents false, then the result of UNPACK (V, MASK = Q, FIELD = M) has the value $\begin{bmatrix} 1 & 2 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}$ and the result of UNPACK (V, MASK = Q, FIELD = 0) has the value $\begin{bmatrix} 0 & 2 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 3 \end{bmatrix}$.

13.11.122 VERIFY (STRING, SET [, BACK, KIND])

Description. Verify that a set of characters contains all the characters in a string by identifying the position of the first character in a string of characters that does not appear in a given set of characters.

Class. Elemental function.

Arguments.

STRING	shall be of type character.
SET	shall be of type character with the same kind type parameter as STRING.
BACK (optional)	shall be of type logical.
KIND (optional)	shall be a scalar integer initialization expression.

Result Characteristics. Integer. If KIND is present, the kind type parameter is that specified by the value of KIND; otherwise the kind type parameter is that of default integer type.

Result Value.

Case (i): If BACK is absent or has the value false and if STRING contains at least one character that is not in SET, the value of the result is the position of the leftmost character of STRING that is not in SET.

Case (ii): If BACK is present with the value true and if STRING contains at least one character that is not in SET, the value of the result is the position of the rightmost character of STRING that is not in SET.

Case (iii): The value of the result is zero if each character in STRING is in SET or if STRING has zero length.

Examples.

Case (i): VERIFY ('ABBA', 'A') has the value 2.

Case (ii): VERIFY ('ABBA', 'A', BACK = .TRUE.) has the value 3.

Case (iii): VERIFY ('ABBA', 'AB') has the value 0.

13.12 The ISO_FORTRAN_ENV intrinsic module

The intrinsic module ISO_FORTRAN_ENV provides public entities relating to the Fortran environment. A processor may provide other public entities in the ISO_FORTRAN_ENV intrinsic module in addition to those listed here.

NOTE 13.20

To avoid potential name conflicts with program entities, it is recommended that a program use the ONLY option in any USE statement that accesses the ISO_FORTRAN_ENV intrinsic module.

13.12.1 Standard input/output units

The processor shall provide three constants giving processor-dependent values for preconnected units (9.4).

13.12.1.1 INPUT_UNIT

The value of the default integer scalar constant INPUT_UNIT identifies the same processor-dependent preconnected external unit as the one identified by an asterisk in a READ statement. The value shall not be -1.

13.12.1.2 OUTPUT_UNIT

The value of the default integer scalar constant OUTPUT_UNIT identifies the same processor-dependent preconnected external unit as the one identified by an asterisk in a WRITE statement. The value shall not be -1.

13.12.1.3 ERROR_UNIT

The value of the default integer scalar constant ERROR_UNIT identifies the processor-dependent preconnected external unit used for the purpose of error reporting. This unit may be the same as OUTPUT_UNIT. The value shall not be -1.

13.12.2 Input/output status

The processor shall provide two constants giving processor-dependent values for end-of-file and end-of-record input/output status (9.9.1). It shall also provide a derived type used to indicate input/output modes.

13.12.2.1 IOSTAT_END

The value of the default integer scalar constant IOSTAT_END is assigned to the variable specified in an IOSTAT= specifier if an end-of-file condition occurs during execution of an input/output statement and no error condition occurs. This value shall be negative.

13.12.2.2 IOSTAT_EOR

The value of the default integer scalar constant IOSTAT_EOR is assigned to the variable specified in an IOSTAT= specifier if an end-of-record condition occurs during execution of an input/output statement and no end-of-file or error condition occurs. This value shall be negative and different from the value of IOSTAT_END.