Numbers

Numerical (Data) Types   To work properly with most FaceWare modules, a compiler must support numbers based on the IEEE or SANE format.   This requirement is met for all of the compilers that we directly support, but each language has its own way of naming these numerical types that does not always make clear the number of bytes used by each type.   Moreover, different compilers supporting the same language may not even agree on type names!   The following table should help prevent any confusion.   The numbers 1 to 12 are sometimes passed to FaceWare modules to indicate the "data type" of variables.

Numerical Type	C	Fortran	Pascal
1. 1-byte integer	char	integer*1	signedByte
2. 2-byte integer	short	integer*2	integer
3. 4-byte integer	long	integer*4	longint
4. 8-byte integer	comp	comp(AF,LF)	computational
5. 4-byte real	float	real*4	real
6. 8-byte real	double(WC)	real*8	double
	short double(LC)		
7. 10-byte real	extended(WC)	real*10(LF)	extended
(w/o 68881/2)	long double(LC..if "Native FP" option on)		
8. 12-byte real	extended(WC)	real*12(AF,LF)	extended
(w/ 68881/2)	long double(LC..if "Native FP" option on)		
12.12-byte real	long double(LC..if "Native FP" option off)		

Array Types   Some modules require programmers to specify the data type of arrays.   This data type is a combination of one of the integers 1 to 12 from the above table, plus a "block type" given by the sign of this integer.   The block type refers to how the numbers in a two-dimensional array are arranged in memory relative to "rows" and "columns" displayed or used by the module.   Positive block types (+1 to +12) indicate that the array consists of blocks of rows (R1C1, R1C2, R1C3... are contiguous), and negative block types (-1 to -12) indicate that the array consists of blocks of columns (R1C1, R2C1, R3C1... are contiguous in memory).

Determining the proper block type (the sign of the data type) to pass to a module depends on which index of the 2-dimensional array corresponds to rows vs. columns, and the language in use.   For example, given a real*8 Fortran array dimensioned as "myArray(10,5)" and an equivalent C or Pascal array also dimensioned as "myArray[10,5]", the data type passed to a module could be either ±6:

non-Fortran "myArray[10,5]" = 10 blocks of 5 numbers each:

- use "+6" to denote 10 rows of 5 columns each
- use "-6" to denote 10 columns of 5 rows each

vs. Fortran "myArray(10,5)" = 5 blocks of 10 numbers each:

- use "+6" to denote 5 rows of 10 columns each
- use "-6" to denote 5 columns of 10 rows each

Type Conversion   Another number-related difference between compilers is the degree to which a compiler will or will not perform automatic type conversions.   Two generalizations can be made:

1. A type conversion will usually be automatically made across an assignment statement between different types of integers or reals (or at least you are warned if the compiler won't make the conversion!).   For example, a 2-byte integer can be assigned to a 4-byte integer.
2. Type conversions are generally not automatically made for the arguments passed in

calls to subroutines, functions, and procedures.   Moreover, a compiler may not warn you of such a type mismatch, leading to some rather bizarre runtime errors.   The major exceptions to this rule are the Pascal compilers which generally allow, for example, 2-byte integers to be passed to 4-byte integers and vice versa.

To solve your type-mismatch problems, most compilers support the use of type-casts or functions that convert one type to another.