Cheat Sheet - Types

This sheet is intended to provide an overview of Fortran types and kinds

01 Base Types

- INTEGER - Represents an Integer number
- REAL - Represents a floating point number. Almost always an IEEE754 floating point number
- COMPLEX - Represents a complex number with real and imaginary parts represented by floating point numbers
- LOGICAL - Represents a Boolean True/False statement
- CHARACTER - Represents a character string. Unlike C/C++ **char** variables these cannot be used as integers. They are just for character strings

02 KINDs

- A kind is an optional INTEGER parameter describing the size or storage format of a type. If no KIND is specified for a type then it is defined as being of "Default Kind". The Fortran standard defines very little about the properties of default kinds but they are usually "sensible" in some way
- INTEGER
  - SELECTED_INT_KIND(r) - Kind that can hold values of at least 10^r
  - Standard 8, 16, 32, and 64 bit integer are defined by
    ```
    INTEGER, PARAMETER :: INT8 = SELECTED_INT_KIND(2)
    INTEGER, PARAMETER :: INT16 = SELECTED_INT_KIND(4)
    INTEGER, PARAMETER :: INT32 = SELECTED_INT_KIND(9)
    INTEGER, PARAMETER :: INT64 = SELECTED_INT_KIND(15)
    ```
  - Same kind parameters defined in ISO_FORTRAN_ENV module in F2008
  - ```
    INTEGER, PARAMETER :: ikind = SELECTED_INT_KIND(10)
    INTEGER(ikind) :: myint = 100_ikind
    INTEGER(KIND=ikind) :: otherint = 200_ikind
    ```
    ```

- If you request an integer kind that cannot be provided then the kind value is -1. This will prevent your code from compiling so do not specify kinds larger than you need

- REAL
  - SELECTED_REAL_KIND(P, R) - Kind that can hold values with at least P digits and an exponential range of at least R
  - Standard 32, 64 and (128-ish) bit reals are defined by
    INTEGER, PARAMETER :: REAL32 = SELECTED_REAL_KIND(6, 37)
    INTEGER, PARAMETER :: REAL64 = SELECTED_REAL_KIND(15, 307)
    INTEGER, PARAMETER :: REAL128 = SELECTED_REAL_KIND(33, 4931)
  - Same kind parameters are defined in the F2008 ISO_FORTRAN_ENV module
  - INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(6, 37)
    INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(15, 307)
    REAL(sp) :: single_var = 10.0_sp
    REAL(KIND=dp) :: double_var = 10.0_dp
  - If you request an real kind that cannot be provided then the kind value is negative and the specific value tells you why the kind could not be provided. This will prevent your code from compiling so do not specify kinds larger than you need

- COMPLEX
  - Same as for REAL
  - Not strictly required to support all real KINDs as COMPLEX but in practice always do
  - INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(6, 37)
    COMPLEX(sp) :: scomplex = (1.0_sp, -1.0_sp)

- LOGICAL
  - Standard does not require any LOGICAL kinds exist in particular (yet), but does require that the default LOGICAL type exists
  - Lots of compilers support LOGICAL kinds with the number being the size of the logical in bytes as the number but this is not required by the standard

    - Lots of compilers use the default logical as LOGICAL(4)
  - Fortran 2008 requires that the ISO_FORTRAN_ENV module provides a LOGICAL_KINDS array describing all available logical kinds
  - INTEGER, PARAMETER :: lkind = 1 !NOT PORTABLE, USE WITH CARE
    LOGICAL(KIND=lkind) :: mylogical = .TRUE._lkind
- CHARACTER
  - SELECTED_CHAR_KIND('DEFAULT') - Request default character string kind, usually ASCII
    SELECTED_CHAR_KIND('ASCII') - Request character string with ASCII encoding
    SELECTED_CHAR_KIND('ISO_10646') - Request character string with ISO 10646/UCS4/UTF-32 unicode encoding
  - Fortran does not require that compilers support anything other than ASCII and DEFAULT and support is currently very patchy.
  - INTEGER, PARAMETER :: ckind = SELECTED_CHAR_KIND('ASCII')
    CHARACTER(KIND=ckind, LEN=11) :: hworld = ckind_"Hello world"
    - Note that the kind specifier on the string constant goes at the **start** of the string constant

- 03 Character Variables

- Have to specify the length of a string when you defined it
  - CHARACTER(LEN=11) :: hworld = "Hello world"
- Also behaves differently for allocatable character variables
  - CHARACTER(LEN=:), ALLOCATABLE :: hworld
    ALLOCATE(CHARACTER(LEN=11) :: hworld)
    hworld = "Hello World"
- Also different when passing a string of unknown length to a subprogram
  - SUBROUTINE pass_string(string)
      CHARACTER(LEN=*), INTENT(IN) :: string
    END SUBROUTINE pass_string

- If you want to pass a character string into a function with the ALLOCATABLE or POINTER attribute then this becomes
  - SUBROUTINE pass_string(string)
    CHARACTER(LEN=:), ALLOCATABLE, INTENT(IN) :: string
    END SUBROUTINE pass_string
- There are reasons why it works like this but they are even more confusing than just remembering the rules

- 04 Older types and their modern equivalents
  - Standards compliant but old fashioned
    - DOUBLE PRECISION -> REAL(KIND=SELECTED_REAL_KIND(15, 307))
  - Not ever standards compliant but commonly supported and common in existing code
    - DOUBLE COMPLEX -> COMPLEX(KIND=SELECTED_REAL_KIND(15, 307))
    - REAL *4 -> REAL(KIND=SELECTED_REAL_KIND(6, 37)
      REAL *8 -> REAL(KIND=SELECTED_REAL_KIND(15, 307))
    - INTEGER *4 -> INTEGER(KIND = SELECTED_INT_KIND(9))
      INTEGER *8 -> INTEGER(KIND = SELECTED_INT_KIND(18))

- 05 ISO_FORTRAN_ENV and ISO_C_BINDING

  - In Fortran 2008 and newer the ISO_FORTRAN_ENV module defines various useful constants including KIND constants for variables. All KINDs specified will be present on all platforms but will have invalid negative values if the platform doesn't support that KIND

    - INT8/INT16/INT32/INT64 - 8/16/32/64 bit integer
      REAL32/REAL64/REAL128 - 32/64/>64 bit integer (REAL128 is larger than double precision but is often 80 bits rather than 128)

    - INTEGER_KINDS - Array containing all supported integer kinds
      REAL_KINDS - Array containing all supported real kinds
      LOGICAL_KINDS - Array containing all supported logical kinds
      CHARACTER_KINDS - Array containing all supported character kinds

  - In Fortran 2003 and newer the ISO_C_BINDING module defines KIND constants that match to the standard sizes for C variables on that platform. There are a lot of them (http://fortranwiki.org/fortran/show/iso_c_binding) but the core ones are

    - C_INT - INTEGER kind matching C "int"
      C_LONG - INTEGER kind matching C "long"
      C_FLOAT - REAL kind matching C "float"
      C_DOUBLE - REAL kind matching C "double"
      C_CHAR - CHARACTER kind matching C "char"

      - Note that C strings and Fortran strings are quite different. The closest match to a C string in Fortran is
        CHARACTER(LEN=1, KIND=C_CHAR), DIMENSION(100) :: c_string
        for a 100 character string. Note also that C strings have to end with a NULL character. This is also defined in ISO_C_BINDING as the constant C_NULL_CHAR and has to be put in place manually since Fortran doesn't use NULL terminated strings

- 06 One KIND from another

  - You can find the KIND of a variable using the **KIND** function
  - This is useful if you want a variable to always be the same KIND as another variable but don't want to share a kind constant (not a common thing but it can happen if you are working with libraries etc)
  - INTEGER(KIND = SELECTED_INT_KIND(9)) :: int1
    INTEGER(KIND = KIND(int1)) :: int2

- 07 KIND parameters to Intrinsic Functions

  - Most Fortran Intrinsic Functions where the kind of the return value is not easily inferred from the kind of the argument take an optional **KIND** argument that allow you to specify the kind of the return value
    - **SIN** or **MAXVAL** should always return a variable of the same kind as the argument so they do not have KIND parameter. **SIZE** always returns an integer regardless of the type of array that it is used on so it does take a KIND parameter
  - INTEGER, PARAMETER :: si = SELECTED_INT_KIND(9)
    INTEGER, PARAMETER :: li = SELECTED_INT_KIND(18)
    REAL, DIMENSION(100) :: array
    !Print both the sizes of the array and the KIND of the size
    PRINT *, SIZE(array, KIND=si), KIND(SIZE(array, KIND=si))
    PRINT *, SIZE(array, KIND=li), KIND(SIZE(array, KIND=li))