

- * Each intrinsic data type has a kind parameter associated with it which is intended to designate a machine representation.
- * A particular implementation might have three “real” kinds: single, double and quadruple precision.
- * The kind is specified with an integer:
 - * **INTEGER (kind=2) or INTEGER(2)**
 - * BUT the standard does not define what the integer means!!! So kind parameters 1, 2 and 3 might be single, double and quadruple precision on one system, but on a different system the kind parameters 4, 8 and 16 may represent the same thing. (example: **mykinds.f90**)

- * The only requirements are that there must be at least **two** real and complex kinds, and at least **one** kind for the integer, logical and character intrinsic types.
- * The intrinsic functions **selected_int_kind** and **selected_real_kind** may be used to select an appropriate kind for a variable or a named constant.
- * **selected_int_kind(P)**: returns the kind value of the smallest integer type that can represent all values ranging from **-10^P to 10^P** (exclusive). If there is no integer kind that can accommodate this range, **selected_int_kind** returns -1.

- * `selected_real_kind(P, R)`: returns the kind value of a real data type with decimal precision of at least **P** digits and exponent range greater than at least **R**.
- * return value:
 - * **-1** = processor does not support a real data type with a precision $\geq P$.
 - * **-2** = processor does not support a real data type with an exponent range $\geq R$.
 - * **-3** = neither is supported
- * example: `whatkinds.f90`

* **KEY:** put definitions in a module and use this throughout your code!!!

CSU GCM example: kinds.F

```
module kinds
  integer, parameter :: int_kind = kind(1), &
                        log_kind = kind(.true.), &
                        real_kind = selected_real_kind(6), &
                        dbl_kind = selected_real_kind(13)
end module kinds
```

Sample computational: elliptic_solver.F

```
module elliptic_solver
  use kinds
  use physical_parameters
  .
  .
  logical (kind=log_kind),parameter :: l_multigrid = .true.
  integer (kind=int_kind) :: bad_apples,iter,iter_max,n1,n2
  real (kind=dbl_kind), parameter :: rconverge = 1.0E-20_dbl_kind
```

- * Constants may have their kind parameter appended where kind matching is required (e.g., in procedure arguments):

```
call some_routine (1.0_dbl_kind, 45_int_kind, x, y, ...)
```

- * Simple example: **passkinds.f90**

- * And another interesting example: **pi.f90**