Enable the compiler's detection of nonconforming code. 6.25, 6.56, 6.57

Enable bounds checking throughout development of a code. Disable bounds checking during production runs only for program units that are critical for performance. 6.8

Use compiler options where available to enable pointer checking throughout development of a code. Disable pointer checking during production runs only for program units that are critical for performance. 6.14

Use compiler options, where available, to find instances of use of uninitialized variables. 6.22

Use all run-time checks that are available during development. 6.52

Use the processor or other static analysis tools to detect and identify obsolescent or deleted features and replace them by better methods. 6.54, 6.58

Use compiler options when available to detect during execution when an integer value overflows. 6.2

Use a tool to automatically refactor unstructured code. 6.31

Declare all variables and use `implicit none` to enforce this. 6.17, 6.54

Use allocatable arrays where array operations involving differently-sized arrays might occur so the left-hand side array is reallocated as needed. 6.8

Use allocatable objects in preference to pointer objects whenever the facilities of allocatable objects are sufficient. 6.14, 6.33, 6.39

Specify explicit interfaces by placing procedures in modules where the procedure is to be used in more than one scope, or by using internal procedures where the procedure is to be used in one scope only. 6.11, 6.32, 6.53

Supply an explicit interface to specify the external attribute for all external procedures invoked. 6.57

Do not use keywords as names when there is any possibility of confusion. 6.17

Do not reuse a name within a nested scope. 6.20

Replace any function with a side effect by a subroutine so that its place in the sequence of computation is certain. 6.24

Declare a function as pure whenever possible. 6.24

Cover cases that are expected never to occur with a case default clause to ensure that unexpected cases are detected and processed, perhaps emitting an error message. 6.27

Use the if construct or select case construct whenever possible, rather than statements that rely on labels, that is, the arithmetic if and go to statements. 6.28

Perform IO on any given file in one programming language only; consider restricting all IO to one language system only. 6.47

Specify argument intents to allow further checking of argument usage. 6.32

Avoid the use of the intrinsic function `transfer`. 6.53

Include an IOSTAT variable in each IO statement and check its value to ensure no errors occurred. *6.6*

Avoid the use of common and equivalence. 6.53

Avoid sequence types. 6.11

Use coarrays only when communication among images is necessary. 6.61

Avoid the use of the `volatile` attribute. 6.61

Avoid the use of the `sync memory` statement for defining and ordering segments. 6.61

Use collective subroutines whenever possible. 6.61, 6.63

Always use intent specifications for dummy arguments. 6.65

Use procedures from a trusted library to perform calculations where floating-point accuracy is needed. Understand the use of the library procedures and test the diagnostic status values returned to ensure the calculation proceeds as expected. 6.4