Fortran Compiler Bug List

This is a list of known issues with different Fortran compilers. This list is largely confined to Fortran bugs in the sense of non-compliance with the Fortran 2003 standard. Some notable extensions, like fpp, are also covered.

The color coding and layout are intended to help developers find specific bugs in the list, but the descriptions are still in Fortran jargon, and it's hard to pick out particularly important bugs. Developers looking for an overview of major/recent bugs can look at Fortran Compiler News.

Note to editors: As of December 2013, it doesn't seem to be possible to fix the width of a column in Confluence[?]. For now, do word wrapping by hand to prevent the table from growing too wide.

- Compiler Support Information
- Fortran Standard Support
- A note on PGI
- Fortran 95 or earlier
- Fortran 2003
- Fortran 2008
- Extensions and the Preprocessor
- Archive: Issues that have been fixed for a long time now, or were never really bugs to begin with

Compiler Support Information

The following color codes are used to highlight bugs that exist in compilers we still support:

No fix known	Fix exists, but in newer versions than supported	Fix exists and applies to all supported versions	Likely invalid
Red	Yellow	Green	Gray

Compilers currently used in testing CESM2 are given here: https://docs.google.com/spreadsheets/d/15QUqsXD1Z0K_rYNTlykBvjTRt8s0XcQw0cfAj9DZbj0/edit#gid=0

Fortran Standard Support

As of post-CESM1.2, most major features of Fortran 2003 were permitted in CESM code, with the following exceptions (due to inconsistent compiler support at that time):

- Finalization
- Parameterized derived types
- User defined I/O
- IEEE modules (IEEE Infinity and NaN related functionality can be used from shr_infnan_mod in csm_share)

As of that time, there were also a few features that were seriously limited due to compiler bugs:

- Character variables with allocatable length:
  - gfortran has related numerous bugs.
  - gfortran does not allow these variables to be components of a type before version 4.9.
- Object oriented programs and OpenMP:
  - OpenMP only updated to Fortran 2003 in version 4.0 of the standard, and its Fortran 2003 specification is still very incomplete.
  - Rule of thumb: Try to avoid "threadprivate" directives, and the "private" clauses, for objects that use Fortran 2003 features, and for objects with allocatable components attached to them.
    - This is not too serious of a restriction; there should not be major problems with objects that are private by default, e.g. because they are local to a function called inside the parallel region.
    - Code that breaks the above rule may be OK, but should be tested carefully, on multiple compilers, to ensure that there is not an issue with accidentally shared memory or a memory leak.

A note on PGI

pgfortran is responsible for a large fraction of bugs on this list, but a high percentage of these have been fixed over the course of 2014. Developers finding new bugs not listed here are encouraged to look at PGI's release TPRs pages to see if the bug has already been discovered and fixed.

Bugs can be reported by anyone through the PGI forums. Of course you should also add any new bugs here.

Fortran 95 or earlier

Vendor (s)	Bug Description	Versions present
Cray	If a procedure argument is of a derived type with allocatable components, and the argument is intent(out) and optional, then calling the procedure with this argument absent produces a segmentation fault.	?
IBM	XLF considers size-zero variables to be initially undefined, and disallows returning them as function results unless they are assigned. • This causes compilation to fail when using "-qhalt=e" (or CMake; for some reason, CMake seems to add this itself?). • While normally it is indeed an error for code to return an uninitialized variable, for a size-zero variable there is usually no requirement by the standard, and assignment to such a variable does nothing.	14.01.0000.0006 Reported through ALCF (#206161). IBM reports that they will fix this.
Intel	A where statement is vectorized by executing both branches and merging the result according to the mask. Therefore, you cannot rely on the mask to filter out operations that result in out-of-bounds accesses, uses of uninitialized data, or floating point exceptions.	<= 13.1.2 Intel seems to consider this a limitation of their vectorization method. May be fixed in 15?
Intel	If a routine uses an intrinsic function, and then is renamed (via use association) to have the same name as that intrinsic, there's an internal error. Workarounds: 1. Avoid reusing a name that belongs to an intrinsic procedure, except when wrapping that procedure for compatibility. 2. Wrap the function in a generic interface, even if there is only one version. • Forum post: https://software.intel.com/en-us/forums/topic/532181	<= 15.0.0 Reported via CISL and again via forum. Intel reports that a fix will be available "later this year" in 2015. (2019-01-03) Given this vague statement, I'm not moving this to the archive list yet.
Intel	Wrong bounds checking message when passing a section of an allocatable array that has a dimension of size 0, to a routine that accepts it as an explicit-shape dummy argument. • Reproducer for the bug. • Issue reference #6000071765	14-15 (Regression from 13) Reported to Intel via Premier Support account. Is fixed in Intel 17 (which will be released later this year)
PGI	With FMA instructions enabled, runs on bluewaters do not give reproducible answers.	13.1-14.1 Reported to PGI, but no reduced case so far.

Fortran 2003

Vendor (s)	Bug Description	Versions present
GNU	Deferred length (i.e. allocatable length) character variables have issues: • Problems with appending: `foo = foo // "a"` • Cannot be derived type components. • Problems with functions that return deferred length characters. • Multiple problems with arrays of deferred length characters.	<=4.9 As of 4.9, most of the remaining bugs relate to arrays of deferred length characters.
GNU	Unclassifiable statement during compilation when assigning to a character array in a derived type contained in a ASSOCIATE statement See https://gcc.gnu.org/bugzilla/show_bug.cgi?id=82121 Workaround is to reference the character variable directly rather than via associate	7.2.0, 8.0, maybe others
IBM	A user-defined constructor (a generic function with the same name as a type) can be mistaken by the compiler for a reference to a structure constructor, if one of the arguments is an expression involving division. • Code that reproduces the issue. • PMR 29174,122,000	<= XLF 14.1.0000.0009 IBM claims fix will be present in November/ December update. (2019-01-03) Given this vague statement, I'm not moving this to the archive list yet.
IBM	Defining assignment(=) on a derived type with an allocatable component breaks default-initialization for that type, when an object of that type is a component of another type. • Reproducer for the issue. • ALCF support #240993	<= XLF 14.1.0000.0009 Reported through ALCF.
Intel	Segmentation fault when initializing a pointer to a class that does not have any data components, using a user-defined constructor. Reproducer: https://github.com/billsacks/compilerbugs-intel_pointer_empty_class Reported to CISL to reproduce and send to intel; however, as of 12-7-15, they have not yet taken any action on this.	<= 16.0.0 (non-standard, but intel looking at)
NAG	Functions that return allocatable arrays of type character cause corruption on the stack. • Code reproducing the problem.	<=6.0 (edit 1017) Reported to NAG.

PGI	ICE when calling a type-bound procedure through an array of polymorphic objects:
	
	`/usr/local/pgi-pgcc-pgf-20.1/linux86-64-llvm/20.1/share/llvm/bin/llc: error: /usr/local/pgi-pgcc-pgf-20.1/linux86-64-llvm/20.1/share/llvm/bin/llc: /tmp/pgf90PFtg7F9Be42q.ll:1034:16: error: use of undefined type named 'struct.BSS4'`
	`%20 = bitcast %struct.BSS4* @.BSS4 to i8*, !dbg !14930`
	
	The code causing the problem is https://github.com/ESCOMP/CISM-wrapper/blob/9aaa9914d55ca1fd9c9fcad0c9d1a34552b75d6e/source_glc/glc_history.F90; the relevant pieces are:
	
	Declarations:
	
	`type :: history_tape_container`
	`private`
	`class(history_tape_base_type), allocatable :: history_tape`
	`end type history_tape_container`
	
	`type(history_tape_container), allocatable :: history_tapes(:)`
	
	Use (this line triggers the ICE):
	
	`call history_tapes(instance_index)%history_tape%write_history(instance, EClock, initial_history)`
	
	Workaround: make a pointer to the instance, and call the subroutine through the pointer: See https://github.com/ESCOMP/CISM-wrapper/commit/90a60b39d67284539d106cae6c92a1c124488668 ; the key pieces are:
	
	`diff --git a/source_glc/glc_history.F90 b/source_glc/glc_history.F90`
	`index a1868eae..61486b9c 100644`
	`--- a/source_glc/glc_history.F90`
	`+++ b/source_glc/glc_history.F90`
	`@@ -42,7 +42,9 @@ module glc_history`
	`class(history_tape_base_type), allocatable :: history_tape`
	`end type history_tape_container`
	
	`- type(history_tape_container), allocatable :: history_tapes(:)`
	`+ type(history_tape_container), allocatable, target :: history_tapes(:)`
	
	`contains`
	
	`@@ -134,9 +136,18 @@ subroutine glc_history_write(instance_index, instance, EClock, initial_history)`
	`type(glad_instance), intent(inout) :: instance`
	`type(ESMF_Clock), intent(in) :: EClock`
	`logical, intent(in), optional :: initial_history`
	`+`
	`+ class(history_tape_base_type), pointer :: htape_ptr`
	`!-----------------------------------------------------------------------`
	
	`- call history_tapes(instance_index)%history_tape%write_history(instance, EClock, initial_history)`
	`+ htape_ptr => history_tapes(instance_index)%history_tape`
	`+ call htape_ptr%write_history(instance, EClock, initial_history)`
	
	`end subroutine glc_history_write`
		Observed in 20.1

GNU Intel PGI	Several compilers don't yet implement allocatable array components as specified in OpenMP 4.0. • Types with allocatable array components were introduced in a TR before Fortran 2003, and they have behavior defined by OpenMP 4.0, but not all compilers have this working. • The main issue is having them in a private clause. If they are local to a function that is called in a parallel region, that is OK, and if they are shared, or in a shared array, that's also OK. • PGI 14.1 seems to have this fixed by the time this problem was discovered. • Intel 14.0.2 report: https://software.intel.com/en-us/forums/topic/509744 (Intel OpenMP still working on Jun/8/'16) • GCC report: http://gcc.gnu.org/bugzilla/show_bug.cgi?id=60928	Various. PGI 14.1 fixed. GCC 4.9.1 fixed. (2019-01-03) Remaining outstanding question is Intel

GNU Intel NAG PGI	Some features are commonly missing from Fortran compilers: • Parameterized derived types (user-defined). • Derived-type I/O	Various. PGI claims support for these features, but this should be regarded as experimental. NAG has derived type kind now (6.0) and len (6.1). Derived-type I/O will be in 6.2 (2017) parameterized derived types for Intel appear in 15 Intel fixed several bugs with derived-type I/O in 17.

Fortran 2008

Vendor (s)	Bug Description	Versions present
GNU	If a dummy argument has the "contiguous" attribute, and the actual argument passed is not contiguous, the data is properly packed into a temporary, but the compiler incorrectly frees the temporary, resulting in a run-time crash. • https://gcc.gnu.org/bugzilla/show_bug.cgi?id=56789	<=4.9 Bug already reported to GCC.
PGI	The intrinsics "erf" and "erfc" are actually external functions that are automatically linked in, rather than intrinsics. This distinction matters mostly because it means that they are not generic and do not have an explicit interface. If you use "erf" for a double precision variable, rather than "derf", you can silently get the wrong version. • Workaround: always use shr_spfn_erf and shr_spfn_erfc; this is necessary anyway in order to compile with NAG versions that lack these functions entirely.	<=12.5 Not fixed in any known version.
Intel	write statements in OpenMP threaded regions to a file opened with the "newunit" specifier can hang (even if the opening of the file is done outside of a threaded region) See https://github.com/ESCOMP/CTSM/issues/1331 for details	< 19.1.1 Appears to be fixed in 19.1.1

Extensions and the Preprocessor

Vendor (s)	Bug Description	Versions present

GNU	Specific intrinsics, such as "sin" and "dcos", do not have the "pure" attribute even if the generic intrinsic is pure. The Fortran standard does not specify this behavior, and in any case, the standards committee is considering deprecating the specific intrinsics in 2015.	<=4. 8 Bug report filed.
GNU	Instead of a dedicated fpp, GNU uses a "traditional" C preprocessor (i.e. one with behavior similar to preprocessors that existed before the standards of the 90's). Some implications: • Unlike with other Fortran preprocessors, "&" is not recognized as a line continuation character. Unfortunately, this means that there is no portable way to do line continuation within a function macro. • Other Fortran-isms (e.g. ".lt." as a synonym for "<") are not portable. • It may be difficult to predict how special characters will be treated within macros. • It takes finesse to do token concatenation.	<=4.9

Archive: Issues that have been fixed for a long time now, or were never really bugs to begin with

Vendor (s)	Bug Description	Versions present
IBM	~~XLF sometimes experiences a run-time error when allocating arrays with OpenMP enabled.~~	14.01.0000.0007 Probably stack size issue, not a bug.
Intel	Wrong results due to inlining sum or product called on certain array sections. • Only visible with -O2 or higher; -O1 gives correct results. • This one is difficult to avoid, except by only calling sum/product on whole dimensions of an array at a time, or verifying results in some way (e.g. using unit tests). • Reproducer for the bug.	13.0.1 - 14.0.2 Fixed in version 15. Not present in Intel 12
Intel	Optimization issue causes the compiler to produce wrong code for certain loops. • Workaround: Compile with reduced optimization (no higher than "-O"), or reorder loops to avoid the bug. • Affected CAM-FV when using compiler version 14 or later. • Reproducer for the bug.	13-15.0.2 (Regression from 12) Reported via CISL by Jim Edwards. Intel reports a fix in version 15.0.3
Intel	In some cases with OpenMP and optimization enabled, setting an entire array to a scalar value can fail, e.g. the last row in the array is not properly set.	<=15.0.0 Appears fixed in 15.0.1
NAG	There are some bugs having to do with deallocating a length zero pointer, which show up if using nagfor's memory tracing features or OpenMP.	<= 5.3.1 NAG believes this to be fixed in 6.0

PGI	One or more bugs related to the location of use statements. The compiler may emit spurious errors about undefined variables unless you move use statements around (from module to procedure level, or vice versa). • These errors are hard to predict, but as a rule of thumb, try to order use statements in a module such that lower level modules precede higher level ones. • Report: http://www.pgroup.com/userforum/viewtopic.php?p=16742#16742 • PGI TPR is #20566.	<= 14.1 Fixed in 14.7
PGI	If a module has the "private" attribute, and it uses a generic interface block to define a specific interface with the same name as the generic, some uses may be unable to resolve the procedure reference. • Report: http://www.pgroup.com/userforum/viewtopic.php?p=16742#16742 • PGI TPR is #20565.	<= 14.1 Fixed in 14.7
PGI	Pointers are sometimes modified when they shouldn't be? • This is a placeholder for a bug that is not very well understood yet. • Detected with a certain version of CAM's tracer_data.F90 • Code in question involves a pointer component that should never be modified (passed as intent(in) to a function and otherwise unused), a type with several pointers that are allocated and deallocated by functions that are called, and a bare pointer used as input. Somehow the first pointer is being changed to point to the last, even though they appear in no statements together. This occurs some time during the first two loop iterations. • Sole known instance of this bug goes away if you add print statements in certain locations, and has not been reproduced with a reduced test case.	<= 14.9 tracer_data case was fixed in 14.10

PGI	Pointer array components of a derived type seem to randomly be resized to 0 when they are passed into a subroutine. As in: call foo(some_vars%bar(bounds%begc:bounds%endc, :)). Within foo, the corresponding dummy argument then appears to be size 0. A number of incarnations of this problem has been observed in CLM4.5 after the big refactor in r081. Initial problems were seen with pgi 13.9, when using the mpi-serial library. However, this was later observed without mpi-serial. Sometimes the problem only occurs when building threaded. Basically, it seems that any seemingly minor change to the build can push the compiler into a situation where this problem occurs in seemingly-random places. A similar problem was observed with a subroutine argument (an assumed-shape array), which was passed on to another subroutine. The actual argument was a pointer component of a derived type (a filter in CLM). A similar problem was observed in CLM when the actual argument was obtained from a function call, as in: call foo(..., cnveg_inst%get_some_var(...)). Workarounds: • Often it works to either insert a print statement before the subroutine call (e.g., writing the ubound of the offending variable before making the subroutine call), or assigning the ubound of the offending variable to a temporary, as in: dummy_to_make_pgi_happy = ubound(somearr, 1). • In one case (UpdateAccVars in CLM's CNDVType.F90), the standard workaround (printing or assigning the ubound of the offending array) did not work. In this case, the workaround that worked was to declare the dummy argument as a pointer (note that the actual argument was already a pointer). • For the case where the actual argument was obtained from a function call, the workaround was to assign the result of the function call to a temporary variable, as in: some_var(:) = cnveg_inst%get_some_var(...) call foo(..., some_var) Note that using pgi 14.10 allows some of the previously-failing CLM tests to pass, but also causes failures in new places (e.g., ERS_D_Lm20.1x1_smallvilleIA.ICLM45BGCCROP.yellowstone_pgi now fails on line 278 of SoilBiogeochemNitrogenStateType.F90, which is an assert on the size of the decomp_cpools_col argument).	<= 14.10 Reported to PGI. No clear reproducer now in 15.1? Also observed in 15.10 reported fixed in 16.5
PGI	Some installations of PGI seem to have excessive stack usage when run on certain files, triggering an internal compiler error. The only known workaround is to attempt to isolate sections of the code that cause the problem, and make pointless changes until it consistently compiles (almost pure trial-and-error). Luckily this bug seems rare.	<= 13.7 No longer reproducable. Assumed fixed.
PGI	Internal compiler error when compiling CLM. • Exact cause is uncertain, but this may represent a problem with the .mod files produced for certain modules, e.g. TemperatureType.	>= 14.1 Fixed in 14.10
GNU	The IEEE intrinsic modules are not implemented. • Ticket here: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=29383	<=4.9 Fixed in 5.1
GNU	Many polymorphism bugs, especially "select type" and "transfer" issues.	Various. Most fixed by 4.8

GNU	Problems accessing allocatable arrays within a polymorphic array. See https://gcc.gnu.org/bugzilla/show_bug.cgi?id=58043 Workaround: if possible, declare the offending variable using 'type' rather than 'class'	<= 4.9. Bug report filed
GNU	Import statement does not work properly with entities renamed via use statement. Workaround is to use an unqualified import statement.	4.6 - 4.7 only. Fixed in 4.8
IBM	~~Polymorphic pointers may lose information about their subclass in certain situations, preventing this information from~~ ~~being discoverable through "select type" (i.e. run-time type identification). This is extremely unpredictable.~~ ~~Here is an example where introducing an extra pointer copy loses information about the dynamic type.~~ ~~Workarounds:~~ • ~~Avoid using run-time type identification (preferred, as it is rarely necessary).~~ • ~~Implement your own run-time type identification (e.g. by providing methods to query dynamic type).~~ • ~~Rearrange code and add select type constructs until the code works (i.e. pure trial and error).~~	<=XLF 14.1.0000.0006 Turned out to be an error due to wrong intent in the test case.
IBM	Internal compiler error when assigning the output of the "eoshift" intrinsic directly to an allocatable array. This only applies when using xlf2003. Workaround is to replace the array on the left side with a section of itself. • E.g. replace `foo = eoshift(...)` with `foo(:) = eoshift(...)`	XLF 14.1.0000.0004 Fixed in XLF 14.1.0000.0006 (In practice, this is the 2013-11 build on Mira.)
IBM	Code involving procedure pointers can sometimes jump to a bad location, leading to illegal instruction errors. • This error was never isolated well enough to report, but the case that triggered it no longer does so on recent versions of the compiler on Mira, suggesting that it was fixed by an update.	XLF 14.1.0000.0006 Tentatively seems fixed in 14.1.0000.0009
Intel	Valid uses of user-defined constructors are rejected by the compiler, if they are called on the outputs of functions (i.e. on "rvalues"). • This occurs regardless of what the contents of the type are. Any function will experience this bug if it is called using a generic name that's the same as the name of a type.	<= 13.0.1 Fixed in 13.1.2
Intel	Internal compiler error when assigning the output of the "merge" intrinsic to an allocatable array, if the option "-assume realloc_lhs" is passed to the compiler. (This option is necessary for full Fortran 2003 support.) • In Intel 12, this bug seems to be triggered by all assignments from "merge". • In Intel 13, this bug seems to be triggered only by assignments where some of the arguments to "merge" are scalars. Workarounds: • Replace array on the left hand side with a section (see the "eoshift" bug for IBM). • Replace the line containing the "merge" call with a "where" construct, or a loop surrounding an "if" construct.	<=13.1.2 Fixed in 14.0.1
Intel	Internal compiler error when re-exporting a derived type with a user-defined constructor, and an operator on that type. • E.g., if you create a type called foo_type, and a generic function called foo_type to construct that type, and define operator(+) to act on foo_type, it can cause the error. • This only occurs when re-exporting, e.g. if public module mod1 uses foo_type and operator(+), and then some other module uses mod1.	<=13.1.2 Fixed in 14.0.1

Intel	Internal compiler error when using a structure constructor inside a user-defined constructor, in certain conditions. • For instance, say you have a type called "foo" that only contains an integer array. Then the following code does not compile: ◦ type(foo) function new_foo(b, c) integer :: b(:), c(:) new_foo = foo(b + c) end function new_foo • Report: http://software.intel.com/en-us/forums/topic/508943	<=14.0.2 Fixed in version 15.
Intel PGI	Internal compiler error when using a structure constructor to produce a type with a procedure pointer. • For Intel, applies only if the pointer is to a function (not subroutine), and only to calls in the same module as where the type is defined. • For PGI, a spurious warning for functions, an internal compiler error for subroutines. • Intel report: http://software.intel.com/en-us/forums/topic/508945 • PGI report (see numbers 4 and 5): http://www.pgroup.com/userforum/viewtopic.php?t=4273	<=15.0.0 (Intel) 14.1 (PGI) Reported through Intel and PGI forums. PGI fixed in 14.7 Intel fixed in 15.0.1
Intel PGI	Spurious error when using a structure constructor directly on an array section (or the output of some intrinsics). For PGI, this is only a warning. • Applies only when the type being constructed is an extension of some other type. • Assuming that the type is foo, with a 1D integer array component, a1 is a 1D array, and a2 is 2D: ◦ This is OK: b = foo(a1) ◦ This is OK: b = foo(a2(:,1) + 0) ◦ This triggers the error: b = foo(a2(:,1)) ◦ This triggers a similar error on Intel: b = foo(spread(1, 1, 5)) • Intel report: http://software.intel.com/en-us/forums/topic/508951 • PGI report (see number 3): http://www.pgroup.com/userforum/viewtopic.php?t=4273	14.0.2 (Intel) 14.1 (PGI) PGI fixed in 14.7 Intel fixed in 15.0.0
Intel	Stack corruption when allocating a polymorphic variable under certain specific circumstances. • This seems to be related to pointers to functions that return a certain kind of derived type. • For details see the report: http://software.intel.com/en-us/forums/topic/508990	<=14.0.2 Fixed in 15.0.0
Intel	Internal compiler error when calling a generic procedure on a component of a derived type, under some conditions. i.e., if bar_type contains a generic procedure do_stuff, then this could cause an internal compiler error: call foo%bar%do_stuff I haven't determined the exact conditions when this problem occurs and doesn't occur. Workaround: call the specific version of the procedure rather than using the generic.	<= 13.1.2. Fixed in 14.0.2
Intel	Internal compiler error when assigning the output of max/min to an entire allocatable array, if one of the arguments of max or min is a scalar. • This bug occurs only when the compiler flag "-assume realloc_lhs" is set, but CESM does set this. • A workaround is to make the left-hand side an array section, i.e. "foo(:) = max(bar, 2)". The "(:)" doesn't do anything, except to work around the bug.	<=13.0.1 Fixed in 13.1.2
Intel	Internal compiler error when using a structure constructor to create any extended type, if the base types are not in scope. • The simplest workaround is to make sure that all base types are in scope where the constructor is called.	Intel 13 only. Fixed in Intel 14. Intel 12 did not have this issue.

Intel	Segmentation fault when doing intrinsic assignment on a derived type variable, which has a component that is a derived type with defined assignment. • Bug report: https://software.intel.com/en-us/forums/topic/532429	<=15.0.0 Reported to Intel via forums. Fixed in Intel 16. And fixed in Intel 15 version from Oct/2014.
NAG	Error when using pointer bounds remapping to set a pointer component of a derived type. Workaround is to remap to a temporary pointer, then set the pointer component from the temporary.	<=5.3.1 Fixed in 6.0
NAG	If you have a function without a result clause, and you reference the name of the output variable in an associate construct, the name is erroneously treated as referring to the function itself rather than the output variable.	<=5.3.1 Fixed in 6.0
NAG	NAG seems to be the last compiler to implement namelist I/O for allocatable/pointer arrays.	<=5.3.1 Implemented in 6.0
NAG	Internal compiler error when calling a generic type-bound function inside an implied do loop. • Code to reproduce the error.	<=6.0 (edit 1017) Fixed in edit 1019
PGI	Allocate with "source=" set to a scalar does not work.	<=13.7 Fixed in 13.9.
PGI	Cannot create a literal empty array (syntax in Fortran 2003 is `[integer::]`). Workaround is to define a function that returns an empty array of the desired type.	<=13.7 Fixed in 13.9
PGI	Compile-time errors when using a structure constructor for a derived type with default initialization. • Normally, code is erroneously rejected with a message about missing constructor arguments. • If a structure constructor has no arguments, the error message is instead about an empty type.	<=11.5 Fixed in 12.5
PGI	Compile-time error when using a structure constructor for a derived type with a type-bound procedure (i.e. a method).	<=12.5 Fixed in 12.10
PGI	Link-time error when using a structure constructor for a derived type with a method.	<=12.10 Fixed in 13.3
PGI	Scope of type-bound procedures is not handled correctly, causing clashes between method names and names in module scope. • This bug seems to have multiple incarnations. • Report for 14.3: http://www.pgroup.com/userforum/viewtopic.php?t=4285	<=14.3 Fixed in 14.7
PGI	When using intrinsic assignment to copy an object with an allocated polymorphic component, dynamic type information is lost.	<=13.9 Fixed in 13.10
PGI	The compiler erroneously rejects code that attempts to allocate non-polymorphic variables of derived type, when the type has no components. The message complains that the type is empty, but Fortran allows this. Workarounds: • Make the variable polymorphic (`class(foo)` instead of `type(foo)`). However, this causes a problem if you use a structure constructor with "source=". Example: "allocate(bar, source=foo())" • Add a meaningless, unused component to the type definition. • Report: http://www.pgroup.com/userforum/viewtopic.php?p=16683#16683	<=13.10 Fixed in 14.7

PGI	Intrinsic assignment sometimes fails to work properly for variables of derived type, when the derived type contains a pointer component and an allocatable component. In this case, the variable on the left-hand side of the assignment ends up with uninitialized values for all of its components (even simple scalar components). Workarounds: • Change the allocatable derived type component to a pointer - this is probably the easiest workaround • It sometimes works to rearrange the order of the derived type components in the derived type declaration • Otherwise, it seems to work to define your own assignment operator for objects of that derived type	<=13.9 PGI claims a fix in 14.2
PGI	PGI encounters an internal error sometimes, when programs use types with type-bound operators (e.g. operator (+)) from a module. • This seems to have something to do with creating unnamed temporary objects. • Typical error messages include "mkexpr1: bad id" and "sym_of_ast: unexpected ast". • Report: http://www.pgroup.com/userforum/viewtopic.php?t=4219	<=14.1 Fixed in 14.7
PGI	PGI does not yet support using the "associate" statement on expressions (rvalues) of arrays. Array elements are usually fine, but other expressions (e.g. those involving arithmetic) do not seem to work yet. • TPR FS#20727	<=14.7, and ==15.1 Fixed in 14.9 Broken again in 15.1 Fixed again in 15.4
PGI	Using the "associate" statement on array subsections causes an internal compiler error ("lowering error").	<=13.9 Fixed as of 14.1
PGI	PGI has an internal "Lowering Error" for some code that uses intrinsics to set array sizes/bounds. • This is triggered by using "size" on an argument to define the size of a function's result, for a function in an interface block, which is then used to specify the interface of a deferred binding. • A similar bug involving the use of "-i8" was fixed in 14.3. • Report (see first issue): http://www.pgroup.com/userforum/viewtopic.php?p=16706#16706	<=14.3 Fixed in 14.7
PGI	PGI has an internal compiler error if you forget the "nopass" attribute on a procedure pointer/binding. • In this case, the code is invalid anyway. • Report (see number 1): http://www.pgroup.com/userforum/viewtopic.php?t=4273	14.1 Fixed in 14.7
PGI	PGI rejects valid code that uses a structure constructor for a polymorphic type that inherits a procedure binding. • Report (see number 2): http://www.pgroup.com/userforum/viewtopic.php?t=4273	14.1 Fixed in 14.7
PGI	Calling a function from a function pointer bound to a type, when the pointer is default-initialized to null(), causes an internal compiler error. • Report (see number 4): http://www.pgroup.com/userforum/viewtopic.php?t=4273	14.1 Fixed in 14.7
PGI	PGI has an internal compiler error if you import a type into an interface earlier in the file than you define the type. • In this case the code is invalid anyway for a compiler that does not look ahead for type definitions (e.g. NAG rejects it). • Report (see number 6): http://www.pgroup.com/userforum/viewtopic.php?p=16675#16675	14.1 Fixed in 14.3
PGI	PGI does not accept "select type" for distinguishing a type with a name that starts with "record". • Report (see number 7): http://www.pgroup.com/userforum/viewtopic.php?p=16676#16676	14.1 Fixed in 14.7

PGI	PGI does not accept a generic binding on a type that aliases only one specific procedure, if that procedure is deferred, if the module does not have a "contains" in it. • Report (see number 8): http://www.pgroup.com/userforum/viewtopic.php?p=16678#16678	14.1 Fixed in 14.7
PGI	PGI produces code with a linking error, if compiling a type that has a polymorphic component of one of its own classes, and does certain things with that type, in a module with a save statement. • Report (see number 9): http://www.pgroup.com/userforum/viewtopic.php?p=16678#16678	14.1 Fixed in 14.7
PGI	The compiler does not recognize that two procedures have compatible interfaces, if one is defined in an interface block and the other is a module procedure, if the procedures accept a procedure with an explicit interface. • Only triggers when "procedure(interface_name)" is used, not with an interface block local to the procedure. • Report (see second issue): http://www.pgroup.com/userforum/viewtopic.php?p=16706#16706	14.1 Fixed in 14.7
PGI	Overridden methods are broken (at least for types with multiple methods). • If you override more than one method of a type, and call the method, at run-time you end up entering the wrong procedure. • Original report from the forums: http://www.pgroup.com/userforum/viewtopic.php?t=3883	<=13.6 Fixed in 13.7.
PGI	pgfortran rejects apparently valid code by saying that it cannot invoke an abstract interface, giving the name of a type- bound procedure that has been overridden with a concrete implementation and which is not actually being called in the given location. It is not clear what triggers this behavior. • Reproducer is pFUnit 3.	13.9-14.10 pFUnit builds in 15.1
PGI	pgfortran rejects apparently valid code by saying that a particular type-bound procedure has not been explicitly declared, despite the fact that (1) the type-bound procedure is declared in the given derived type, and (2) that type-bound procedure is not actually being called in the file that generates the error. In the one observed instance, the file giving the error referenced many types that all had the same names for type-bound procedures (e.g., Init). • A workaround was to rename these methods in the newly-introduced type (e.g., rename Init to IrrigationInit in that type). • TPR FS#21165	<= 14.10 Fixed in 15.1
PGI	Case where method calls could only be resolved if the methods had *not* been renamed to something different from the original procedure's name. • TPR FS#21166	<=14.10 Fixed in version 15 (official release will be 15.1)
PGI	pgfortran encounters a segfault when compiling CLM's filterMod, if the flag "-Mallocatable=03" is given (needed for full Fortran 2003 compliance). • This has been reproduced with a smaller case involving two stubs and a slightly modified version of clmtype.F90. • This probably has something to do with the large number of derived type pointer components in clmtype, since removing enough of these types (regardless of which ones?) will get rid of the bug. • TPR FS#20758	<=14.7 Fixed in 14.9

PGI	pgfortran 14.7 has an internal compiler error when compiling the file in the attached tarball. • Based on the line number cited, this seems to have something to do with using move_alloc to transfer an allocatable component from one object to another. • TPR FS#20796 • The tarball.	<=14.7 Fixed in 14.9
PGI	pgfortran 14 fails to look recursively through subobjects of a derived type object when doing automatic deallocation. • Reproducer (run with valgrind to see unfreed memory) • TPR FS#21079	<=14.10 Fixed in 15.1
PGI	pgfortran 14 internal compiler error with "-Mallocatable=03" and allocatable components of allocatable components. • Reproducer (only has error with -Mallocatable=03) • TPR FS#21080	<=14.10 Fixed in 15.1
PGI	pgfortran 14 gives a warning if you set the values pointed to by pointer components of intent(in) values, even though the Fortran standard allows this. • Reproducer (compile with -c) • TPR FS#21081	<=14.10 Fixed in 15.1
PGI	pgfortran 14 loses information about variables declared in the child class of a polymorphic entity allocated with "source=". i.e., if you have something with this pattern: type, abstract :: foo_base_type integer, allocatable :: barvar1(:) end type type, extends(foo_base_type) :: foo_concrete_type integer, allocatable :: barvar2(:) end type class(foo_base_type) :: my_foo Then, if the dynamic type of my_foo is foo_concrete_type, my_foo%barvar2 will sometimes get resized to 0, although my_foo%barvar1 is fine. This can be reproduced with: `PET_P15x2_Ly3.f10_f10.ICLM45BGCCROP.yellowstone_pgi.clm-irrig_o3_reduceOutput` run on this tag, although the problem is not actually threading-related. • Reproducer is test_poly_pointers.F90 from this tarball. • TPR FS#21130	<=14.10 Fixed in 15.1
PGI	When not all allocatable fields are allocated in the parent class of a polymorphic entity allocated with "source=", the allocation may cause a segfault. • Reproducer is dumps_core_when_run.F90 from this tarball.	<=14.10. Fixed in PGI 15; the official release will be 15.1

PGI	pgfortran 14 sometimes has trouble with sourced allocation of a polymorphic entity. For example, in this tag many tests, such as ERS_D.f10_f10.ICLM45.yellowstone_pgi.clm-reduceOutput die at runtime in the sourced allocation of ozone_inst, line 287 of models/lnd/clm/src/main/clm_instMod.F90: allocate(ozone_inst, source = create_and_init_ozone_type(bounds)) That allocate line works fine if creating an instance of ozone_type, but NOT if creating an instance of ozone_off_type. The same allocate statement works fine in the n09 tag, where the difference is that n09 has some variables declared in ozone_type rather than in ozone_base_type. • A reduced test case is test_alloc_child in this tarball.	<= 14.10 Fixed in PGI 15; the official release will be 15.1
PGI	pgfortran 15 hangs after a compile time glibc error. • Reproducer is pgi_glibc_error.F90 in this tarball. • TPR FS#21129.	<= 14.10 Fixed in 15.1
PGI	When building CLM code with mpi-serial, using pgi14.7, Bill Sacks encountered the error: PGF90-F-0000-Internal compiler error. normalize_forall_array: non-conformable 37663 (/glade/p/work/sacks/cesm_code/clm_modular_irrigation/models/lnd/clm/src/main/clm_initializeMod.F90: 723) PGF90/x86-64 Linux 14.7-0: compilation aborted This was with this tag e.g., for ERS_Lm3.1x1_smallvilleIA.ICLM45BGCCROP.yellowstone_pgi A workaround was to print the size of the array arguments just before the subroutine call on line 723. • The "normalize_forall_array: non-conformable" internal compiler error also occurs using this tag, e.g., for PET_P15x2_Lm13.f10_f10.IHISTCLM45BGC.yellowstone_pgi.clm-reduceOutput In this case, the problem appeared in clm_driver, in the call to CalcIrrigationNeeded. The workaround in this case was to declare a new subroutine local variable - an allocatable array. (see r65797 on the branch https://svn-ccsm-models.cgd.ucar.edu/clm2/branches/ozone_polymorphism)	<= 14.10 Fixed in 15.1? No known case triggers it anymore.
PGI	Regression in early access version 15.0 of PGI. This should not be necessary to work around because it will probably be fixed by the release. The issue is that association to expressions of the form "bar%x(:)" will not work if "bar%x is an allocatable array. • TPR FS#21192	15.0 only

PGI	pgfortran 14 and 15 give internal compiler errors when setting an allocatable character variable equal to an expression that involves a function result. For example: character(len=:), allocatable :: mystring mystring = 'hello' // two_char_string() results in: PGF90-S-0000-Internal compiler error. string_expr_length: ast not string op 13 Workaround: introduce an intermediate variable, as in: character(len=:), allocatable :: temp_string character(len=:), allocatable :: mystring temp_string = two_char_string() mystring = 'hello' // temp_string Reproducer is test_allocatable_char_broken.F90 in pgi_allocatable_char.tar	<= 15.4 Fixed in 15.7
PGI	POSSIBLE BUG: Allocatable character components of derived types sometimes get filled with garbage? This showed up in a few yellowstone-pgi tests (using pgi 15.10) in https://svn-ccsm-models.cgd.ucar.edu/clm2/branch_tags/product_pools_gridcell2_tags/ product_pools_gridcell2_n02_clm4_5_8_r172 and was fixed in tag product_pools_gridcell2_n03_clm4_5_8_r172 by changing allocatable character variables to fixed-length. Specifically, this is in SpeciesIsotopeType and SpeciesNonIsotopeType. In the n02 tag, I got failures like: ERP_P15x2_D_Ld5.f10_f10.I1850CLM45BGC.yellowstone_pgi.clm-ciso 8: masterlist_addfld ERROR:1od_CROPPROD11 already on list as well as failures that suggested that different tasks were putting different garbage in the field: e.g.: ERI_Ld9.f09_g16.I1850CRUCLM45BGC.yellowstone_pgi.clm-drydepnomegan 451:Abort with message Variable names are defined inconsistently among processes Possibly relevant: the objects of the derived type in question were actually polymorphic entities. I'm not positive that this is a compiler bug - it might have been programmer error. I'm filing it here to keep an eye on it.	<= 15.10 May have been programmer error rather than a compiler bug reported fixed in 16.5

PGI	Adding an interface to subroutines in seq_infodata_mod.F90 caused a compiler error:
	
	`*** glibc detected *** /glade/u/ssg/ys/opt/pgi/15.10/`
	
	`linux86-64/15.10/bin/pgf901: free(): invalid next size (fast): 0x000000000242fb00 ***`
	`recipe for target 'seq_infodata_mod.o' failed gmake: *** [seq_infodata_mod.o] Error 127`
	
	The issue was caused by adding a new interface to seq_infodata_GetData and seq_infodata_PutData. The code
	
	which triggered the PGI compiler error is in the current version of cime/driver_cpl/shr/seq_infodata_mod.F90 in
	
	sections set off by #ifndef CPRPGI.
		15.10 although it
		
		seems to also be in
		
		other versions.
		
		reported fixed in 16.5

PGI Cray	Operators in IEEE_ARITHMETIC are not always elemental/pure when they should be. • For PGI, the operators can be used on scalars, with a spurious warning for pure procedures. • For Cray, using the operators in a pure procedure causes an error	<=13.7 (PGI) Fixed in 13.9 (PGI) Fixed in an unknown version for Cray.

GNU	Finalization is not supported by gfortran yet.	<=4.8 4.9 has preliminary support (at least code with finalization should compile).
PGI	(Fortran 2008) Multidimensional pointer bounds remapping can be confused by lower bounds of an array that are lower than 1. • TPR 19660	<=13.10 Fixed in 14.1
PGI	(Fortran 2008) The compiler claims that the "contiguous" attribute is incompatible with "intent", and rejects valid code in which dummy arguments have both. • Workaround: put "intent" before "contiguous". • Report: http://www.pgroup.com/userforum/viewtopic.php?t=4406	<=14.1 Fixed in 14.7
NAG	(Extensions and the preprocessor) Bugs involving the preprocessor and line length limits. NAG's fpp guarantees that if the input file obeys the 132 character limit, then so will the output file. However, there have been a few bugs that result in: • Two lines being merged together when the first one contains blank characters that exceed the 132 character limit. • Line continuation characters ("&") being inserted in inappropriate locations in lines.	<=5.3.1 Fixed in 6.0
NAG	(Extensions and the preprocessor) Bugs involving function macros given no arguments. If you define a function macro, then refer to the name of that macro without providing arguments (e.g. in comments about the macro), the preprocessor may produce odd output, such as files that are incomplete, or an infinite number of control characters. Workarounds: • Always give a function macro the right number of arguments, even in comments. • Use "-Wp,-macro=no_com" to turn off macro expansion in comments.	<=6.0 (edit 1017) Report filed w/ NAG support. Nag reports a fix in edit 1032.