NAME

gfortran − GNU Fortran compiler

SYNOPSIS

gfortran [**−c**|**−S**|**−E**]
[**−g**] [**−pg**] [**−O***level*]
[**−W***warn*...] [**−pedantic**]
[**−I***dir*...] [**−L***dir*...]
[**−D***macro*[=*defn*]...] [**−U***macro*]
[**−f***option*...]
[**−m***machine-option*...]
[**−o** *outfile*] *infile*...

Only the most useful options are listed here; see below for the remainder.

DESCRIPTION

The **gfortran** command supports all the options supported by the **gcc** command. Only options specific to GNU Fortran are documented here.

All GCC and GNU Fortran options are accepted both by **gfortran** and by **gcc** (as well as any other drivers built at the same time, such as **g++**), since adding GNU Fortran to the GCC distribution enables acceptance of GNU Fortran options by all of the relevant drivers.

In some cases, options have positive and negative forms; the negative form of **−ffoo** would be **−fno−foo**. This manual documents only one of these two forms, whichever one is not the default.

OPTIONS

Here is a summary of all the options specific to GNU Fortran, grouped by type. Explanations are in the following sections.

Fortran Language Options

−fall−intrinsics −fallow−argument−mismatch −fallow−invalid−boz −fbackslash −fcray−pointer −fd−lines−as−code −fd−lines−as−comments −fdec −fdec−char−conversions −fdec−structure −fdec−intrinsic−ints −fdec−static −fdec−math −fdec−include −fdec−format−defaults −fdec−blank−format−item −fdefault−double−8 −fdefault−integer−8 −fdefault−real−8 −fdefault−real−10 −fdefault−real−16 −fdollar−ok −ffixed−line−length−*n*** −ffixed−line−length−none −fpad−source −ffree−form −ffree−line−length−***n*** −ffree−line−length−none −fimplicit−none −finteger−4−integer−8 −fmax−identifier−length −fmodule−private −ffixed−form −fno−range−check −fopenacc −fopenmp −freal−4−real−10 −freal−4−real−16 −freal−4−real−8 −freal−8−real−10 −freal−8−real−16 −freal−8−real−4 −std=***std*** −ftest−forall−temp**

Preprocessing Options

−A−*question*[=*answer*] **−A***question=answer* **−C −CC −D***macro*[=*defn*] **−H −P −U***macro* **−cpp −dD −dI −dM −dN −dU −fworking−directory −imultilib** *dir* **−iprefix** *file* **−iquote −isysroot** *dir* **−isystem** *dir* **−nocpp −nostdinc −undef**

Error and Warning Options

−Waliasing −Wall −Wampersand −Warray−bounds −Wc−binding−type −Wcharacter−truncation −Wconversion −Wdo−subscript −Wfunction−elimination −Wimplicit−interface −Wimplicit−procedure −Wintrinsic−shadow −Wuse−without−only −Wintrinsics−std −Wline−truncation −Wno−align−commons −Wno−overwrite−recursive −Wno−tabs −Wreal−q−constant −Wsurprising −Wunderflow −Wunused−parameter −Wrealloc−lhs −Wrealloc−lhs−all −Wfrontend−loop−interchange −Wtarget−lifetime −fmax−errors=*n*** −fsyntax−only −pedantic −pedantic−errors**

Debugging Options

−fbacktrace −fdump−fortran−optimized −fdump−fortran−original −fdebug−aux−vars −fdump−fortran−global −fdump−parse−tree −ffpe−trap=*list* **−ffpe−summary=***list*

Directory Options
−I*dir* **−J***dir* **−fintrinsic−modules−path** *dir*

Link Options
−static−libgfortran

Runtime Options
−fconvert=*conversion* **−fmax−subrecord−length=***length* **−frecord−marker=***length* **−fsign−zero**

Interoperability Options
−fc−prototypes −fc−prototypes−external

Code Generation Options
−faggressive−function−elimination −fblas−matmul−limit=*n* **−fbounds−check −ftail−call−workaround −ftail−call−workaround=***n* **−fcheck−array−temporaries −fcheck=**<*all*|*array−temps*|*bits*|*bounds*|*do*|*mem*|*pointer*|*recursion*> **−fcoarray=**<*none*|*single*|*lib*> **−fexternal−blas −ff2c −ffrontend−loop−interchange −ffrontend−optimize −finit−character=***n* **−finit−integer=***n* **−finit−local−zero −finit−derived −finit−logical=**<*true*|*false*> **−finit−real=**<*zero*|*inf*|*−inf*|*nan*|*snan*> **−finline−matmul−limit=***n* **−finline−arg−packing −fmax−array−constructor=***n* **−fmax−stack−var−size=***n* **−fno−align−commons −fno−automatic −fno−protect−parens −fno−underscoring −fsecond−underscore −fpack−derived −frealloc−lhs −frecursive −frepack−arrays −fshort−enums −fstack−arrays**

Options controlling Fortran dialect

The following options control the details of the Fortran dialect accepted by the compiler:

−ffree−form
−ffixed−form

Specify the layout used by the source file. The free form layout was introduced in Fortran 90. Fixed form was traditionally used in older Fortran programs. When neither option is specified, the source form is determined by the file extension.

−fall−intrinsics

This option causes all intrinsic procedures (including the GNU-specific extensions) to be accepted. This can be useful with **−std=f95** to force standard-compliance but get access to the full range of intrinsics available with **gfortran**. As a consequence,**−W intrinsics−std** will be ignored and no user-defined procedure with the same name as any intrinsic will be called except when it is explicitly declared `EXTERNAL`.

−fallow−argument−mismatch

Some code contains calls to external procedures with mismatches between the calls and the procedure definition, or with mismatches between different calls. Such code is non-conforming, and will usually be flagged with an error. This options degrades the error to a warning, which can only be disabled by disabling all warnings via **−w**. Only a single occurrence per argument is flagged by this warning. **−fallow−argument−mismatch** is implied by **−std=legacy**.

Using this option is *strongly* discouraged. It is possible to provide standard-conforming code which allows different types of arguments by using an explicit interface and `TYPE(*)`.

−fallow−invalid−boz

A BOZ literal constant can occur in a limited number of contexts in standard conforming Fortran. This option degrades an error condition to a warning, and allows a BOZ literal constant to appear where the Fortran standard would otherwise prohibit its use.

−fd−lines−as−code
−fd−lines−as−comments

Enable special treatment for lines beginning with `d` or `D` in fixed form sources. If the **−fd−lines−as−code** option is given they are treated as if the first column contained a blank. If the **−fd−lines−as−comments** option is given, they are treated as comment lines.

−fdec
DEC compatibility mode. Enables extensions and other features that mimic the default behavior of older compilers (such as DEC). These features are non-standard and should be avoided at all costs. For details on GNU Fortran's implementation of these extensions see the full documentation.

Other flags enabled by this switch are: **−fdollar−ok −fcray−pointer −fdec−char−conversions −fdec−structure −fdec−intrinsic−ints −fdec−static −fdec−math −fdec−include −fdec−blank−format−item −fdec−format−defaults**

If **−fd−lines−as−code**/**−fd−lines−as−comments** are unset, then **−fdec** also sets **−fd−lines−as−comments**.

−fdec−char−conversions
Enable the use of character literals in assignments and `DATA` statements for non-character variables.

−fdec−structure
Enable DEC `STRUCTURE` and `RECORD` as well as `UNION`, `MAP`, and dot ('.') as a member separator (in addition to '%'). This is provided for compatibility only; Fortran 90 derived types should be used instead where possible.

−fdec−intrinsic−ints
Enable B/I/J/K kind variants of existing integer functions (e.g. BIAND, IIAND, JIAND, etc...). For a complete list of intrinsics see the full documentation.

−fdec−math
Enable legacy math intrinsics such as COTAN and degree-valued trigonometric functions (e.g. TAND, ATAND, etc...) for compatability with older code.

−fdec−static
Enable DEC-style STATIC and AUTOMATIC attributes to explicitly specify the storage of variables and other objects.

−fdec−include
Enable parsing of INCLUDE as a statement in addition to parsing it as INCLUDE line. When parsed as INCLUDE statement, INCLUDE does not have to be on a single line and can use line continuations.

−fdec−format−defaults
Enable format specifiers F, G and I to be used without width specifiers, default widths will be used instead.

−fdec−blank−format−item
Enable a blank format item at the end of a format specification i.e. nothing following the final comma.

−fdollar−ok
Allow **$** as a valid non-first character in a symbol name. Symbols that start with **$** are rejected since it is unclear which rules to apply to implicit typing as different vendors implement different rules. Using **$** in `IMPLICIT` statements is also rejected.

−fbackslash
Change the interpretation of backslashes in string literals from a single backslash character to "C−style" escape characters. The following combinations are expanded `\a`, `\b`, `\f`, `\n`, `\r`, `\t`, `\v`, `\\`, and `\0` to the ASCII characters alert, backspace, form feed, newline, carriage return, horizontal tab, vertical tab, backslash, and NUL, respectively. Additionally, `\x`*nn*, `\u`*nnnn* and `\U`*nnnnnnnn* (where each *n* is a hexadecimal digit) are translated into the Unicode characters corresponding to the specified code points. All other combinations of a character preceded by \ are unexpanded.

−fmodule−private
Set the default accessibility of module entities to `PRIVATE`. Use-associated entities will not be accessible unless they are explicitly declared as `PUBLIC`.

−ffixed−line−length−*n*
Set column after which characters are ignored in typical fixed-form lines in the source file, and, unless `−fno−pad−source`, through which spaces are assumed (as if padded to that length) after the ends

of short fixed-form lines.

Popular values for *n* include 72 (the standard and the default), 80 (card image), and 132 (corresponding to ‘‘extended-source’’ options in some popular compilers). *n* may also be **none**, meaning that the entire line is meaningful and that continued character constants never have implicit spaces appended to them to fill out the line. **−ffixed−line−length−0** means the same thing as **−ffixed−line−length−none**.

−fno−pad−source
By default fixed-form lines have spaces assumed (as if padded to that length) after the ends of short fixed-form lines. This is not done either if **−ffixed−line−length−0**, **−ffixed−line−length−none** or if **−fno−pad−source** option is used. With any of those options continued character constants never have implicit spaces appended to them to fill out the line.

−ffree−line−length−*n*
Set column after which characters are ignored in typical free-form lines in the source file. The default value is 132. *n* may be **none**, meaning that the entire line is meaningful. **−ffr ee−line−length−0** means the same thing as **−ffree−line−length−none**.

−fmax−identifier−length=*n*
Specify the maximum allowed identifier length. Typical values are 31 (Fortran 95) and 63 (Fortran 2003 and Fortran 2008).

−fimplicit−none
Specify that no implicit typing is allowed, unless overridden by explicit `IMPLICIT` statements. This is the equivalent of adding `implicit none` to the start of every procedure.

−fcray−pointer
Enable the Cray pointer extension, which provides C−like pointer functionality.

−fopenacc
Enable the OpenACC extensions. This includes OpenACC `!$acc` directives in free form and `c$acc`, `*$acc` and `!$acc` directives in fixed form, `!$` conditional compilation sentinels in free form and `c$`, `*$` and `!$` sentinels in fixed form, and when linking arranges for the OpenACC runtime library to be linked in.

−fopenmp
Enable the OpenMP extensions. This includes OpenMP`!$omp` directi ves in free form and `c$omp`, `*$omp` and `!$omp` directives in fixed form, `!$` conditional compilation sentinels in free form and `c$`, `*$` and `!$` sentinels in fixed form, and when linking arranges for the OpenMP runtime library to be linked in. The option **−fopenmp** implies **−frecursive**.

−fno−range−check
Disable range checking on results of simplification of constant expressions during compilation. For example, GNU Fortran will give an error at compile time when simplifying `a = 1. / 0`. With this option, no error will be given and `a` will be assigned the value `+Infinity`. If an expression evaluates to a value outside of the relevant range of [`−HUGE()`:`HUGE()`], then the expression will be replaced by `−Inf` or `+Inf` as appropriate. Similarly, `DATA i/Z'FFFFFFFF'/` will result in an integer overflow on most systems, but with **−fno−range−check** the value will ‘‘wrap around’’ and `i` will be initialized to −1 instead.

−fdefault−integer−8
Set the default integer and logical types to an 8 byte wide type. This option also affects the kind of integer constants like `42`. Unlike **−finteger−4−integer−8**, it does not promote variables with explicit kind declaration.

−fdefault−real−8
Set the default real type to an 8 byte wide type. This option also affects the kind of non-double real constants like `1.0`. This option promotes the default width of `DOUBLE PRECISION` and double real constants like `1.d0` to 16 bytes if possible. If `−fdefault−double−8` is given along with `fdefault−real−8`, `DOUBLE PRECISION` and double real constants are not promoted. Unlike

−freal−4−real−8, `fdefault−real−8` does not promote variables with explicit kind declarations.

−fdefault−real−10
Set the default real type to an 10 byte wide type. This option also affects the kind of non-double real constants like `1.0`. This option promotes the default width of `DOUBLE PRECISION` and double real constants like `1.d0` to 16 bytes if possible. If `−fdefault−double−8` is given along with `fdefault−real−10`, `DOUBLE PRECISION` and double real constants are not promoted. Unlike **−freal−4−real−10**, `fdefault−real−10` does not promote variables with explicit kind declarations.

−fdefault−real−16
Set the default real type to an 16 byte wide type. This option also affects the kind of non-double real constants like `1.0`. This option promotes the default width of `DOUBLE PRECISION` and double real constants like `1.d0` to 16 bytes if possible. If `−fdefault−double−8` is given along with `fdefault−real−16`, `DOUBLE PRECISION` and double real constants are not promoted. Unlike **−freal−4−real−16**, `fdefault−real−16` does not promote variables with explicit kind declarations.

−fdefault−double−8
Set the `DOUBLE PRECISION` type and double real constants like `1.d0` to an 8 byte wide type. Do nothing if this is already the default. This option prevents **−fdefault−real−8**, **−fdefault−real−10**, and **−fdefault−real−16**, from promoting `DOUBLE PRECISION` and double real constants like `1.d0` to 16 bytes.

−finteger−4−integer−8
Promote all `INTEGER(KIND=4)` entities to an `INTEGER(KIND=8)` entities. If `KIND=8` is unavailable, then an error will be issued. This option should be used with care and may not be suitable for your codes. Areas of possible concern include calls to external procedures, alignment in `EQUIVALENCE` and/or `COMMON`, generic interfaces, BOZ literal constant conversion, and I/O. Inspection of the intermediate representation of the translated Fortran code, produced by **−fdump−tree−original**, is suggested.

−freal−4−real−8
−freal−4−real−10
−freal−4−real−16
−freal−8−real−4
−freal−8−real−10
−freal−8−real−16
Promote all `REAL(KIND=M)` entities to `REAL(KIND=N)` entities. If `REAL(KIND=N)` is unavailable, then an error will be issued. The `−freal−4−` flags also affect the default real kind and the `−freal−8−` flags also the double-precision real kind. All other real-kind types are unaffected by this option. The promotion is also applied to real literal constants of default and double-precision kind and a specified kind number of 4 or 8, respectively. However, `−fdefault−real−8`, `−fdefault−real−10`, `−fdefault−real−10`, and `−fdefault−double−8` take precedence for the default and double-precision real kinds, both for real literal constants and for declarations without a kind number. Note that for`REAL(KIND=KIND(1.0))` the literal may get promoted and then the result may get promoted again. These options should be used with care and may not be suitable for your codes. Areas of possible concern include calls to external procedures, alignment in `EQUIVALENCE` and/or `COMMON`, generic interfaces, BOZ literal constant conversion, and I/O and calls to intrinsic procedures when passing a value to the `kind=` dummy argument. Inspection of the intermediate representation of the translated Fortran code, produced by **−fdump−fortran−original** or **−fdump−tree−original**, is suggested.

−std=*std*
Specify the standard to which the program is expected to conform, which may be one of **f95**, **f2003**, **f2008**, **f2018**, **gnu**, or **legacy**. The default value for *std* is **gnu**, which specifies a superset of the latest Fortran standard that includes all of the extensions supported by GNU Fortran, although warnings will be given for obsolete extensions not recommended for use in new code. The **legacy** value is equivalent but without the warnings for obsolete extensions, and may be useful for old non-standard programs. The**f95**, **f2003**, **f2008**, and **f2018** v alues specify strict conformance to the Fortran 95,

Fortran 2003, Fortran 2008 and Fortran 2018 standards, respectively; errors are given for all extensions beyond the relevant language standard, and warnings are given for the Fortran 77 features that are permitted but obsolescent in later standards. The deprecated option **−std=f2008ts** acts as an alias for **−std=f2018**. It is only present for backwards compatibility with earlier gfortran versions and should not be used any more.

−ftest−forall−temp
Enhance test coverage by forcing most forall assignments to use temporary.

Enable and customize preprocessing
Preprocessor related options. See section **Preprocessing and conditional compilation** for more detailed information on preprocessing in **gfortran**.

−cpp
−nocpp
Enable preprocessing. The preprocessor is automatically invoked if the file extension is *.fpp*, *.FPP*, *.F*, *.FOR*, *.FTN*, *.F90*, *.F95*, *.F03* or *.F08*. Use this option to manually enable preprocessing of any kind of Fortran file.

To disable preprocessing of files with any of the above listed extensions, use the negative form: **−nocpp**.

The preprocessor is run in traditional mode. Any restrictions of the file-format, especially the limits on line length, apply for preprocessed output as well, so it might be advisable to use the **−ffree−line−length−none** or **−ffixed−line−length−none** options.

−dM
Instead of the normal output, generate a list of `'#define'` directives for all the macros defined during the execution of the preprocessor, including predefined macros. This gives you a way of finding out what is predefined in your version of the preprocessor. Assuming you have no file *foo.f90*, the command

```
touch foo.f90; gfortran −cpp −E −dM foo.f90
```

will show all the predefined macros.

−dD
Like **−dM** except in two respects: it does not include the predefined macros, and it outputs both the `#define` directives and the result of preprocessing. Both kinds of output go to the standard output file.

−dN
Like **−dD**, but emit only the macro names, not their expansions.

−dU
Like **dD** except that only macros that are expanded, or whose definedness is tested in preprocessor directives, are output; the output is delayed until the use or test of the macro; and `'#undef'` directives are also output for macros tested but undefined at the time.

−dI
Output `'#include'` directives in addition to the result of preprocessing.

−fworking−directory
Enable generation of linemarkers in the preprocessor output that will let the compiler know the current working directory at the time of preprocessing. When this option is enabled, the preprocessor will emit, after the initial linemarker, a second linemarker with the current working directory followed by two slashes. GCC will use this directory, when it is present in the preprocessed input, as the directory emitted as the current working directory in some debugging information formats. This option is implicitly enabled if debugging information is enabled, but this can be inhibited with the negated form **−fno−working−directory**. If the **−P** flag is present in the command line, this option has no effect, since no `#line` directives are emitted whatsoever.

−idirafter *dir*
Search *dir* for include files, but do it after all directories specified with **−I** and the standard system directories have been exhausted. *dir* is treated as a system include directory. If dir begins with =, then the = will be replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

−imultilib *dir*
Use *dir* as a subdirectory of the directory containing target-specific C++ headers.

−iprefix *prefix*
Specify *prefix* as the prefix for subsequent **−iwithprefix** options. If the *prefix* represents a directory, you should include the final `'/'`.

−isysroot *dir*
This option is like the **−−sysroot** option, but applies only to header files. See the **−−sysroot** option for more information.

−iquote *dir*
Search *dir* only for header files requested with `#include "file"`; they are not searched for `#include <file>`, before all directories specified by **−I** and before the standard system directories. If *dir* begins with =, then the = will be replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

−isystem *dir*
Search *dir* for header files, after all directories specified by **−I** but before the standard system directories. Mark it as a system directory, so that it gets the same special treatment as is applied to the standard system directories. If *dir* begins with =, then the = will be replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

−nostdinc
Do not search the standard system directories for header files. Only the directories you have specified with **−I** options (and the directory of the current file, if appropriate) are searched.

−undef
Do not predefine any system-specific or GCC-specific macros. The standard predefined macros remain defined.

−A*predicate*=*answer*
Make an assertion with the predicate *predicate* and answer *answer*. This form is preferred to the older form −A predicate(answer), which is still supported, because it does not use shell special characters.

−A−*predicate*=*answer*
Cancel an assertion with the predicate *predicate* and answer *answer*.

−C Do not discard comments. All comments are passed through to the output file, except for comments in processed directives, which are deleted along with the directive.

You should be prepared for side effects when using **−C**; it causes the preprocessor to treat comments as tokens in their own right. For example, comments appearing at the start of what would be a directive line have the effect of turning that line into an ordinary source line, since the first token on the line is no longer a `'#'`.

Warning: this currently handles C−Style comments only. The preprocessor does not yet recognize Fortran-style comments.

−CC
Do not discard comments, including during macro expansion. This is like **−C**, except that comments contained within macros are also passed through to the output file where the macro is expanded.

In addition to the side-effects of the **−C** option, the **−CC** option causes all C++−style comments inside a macro to be converted to C−style comments. This is to prevent later use of that macro from inadvertently commenting out the remainder of the source line. The **−CC** option is generally used to support lint comments.

Warning: this currently handles C−− and C++−−Style comments only. The preprocessor does not yet recognize Fortran-style comments.

−D*name*
Predefine name as a macro, with definition `1`.

−D*name*=*definition*
The contents of *definition* are tokenized and processed as if they appeared during translation phase three in a `'#define'` directive. In particular, the definition will be truncated by embedded newline characters.

If you are invoking the preprocessor from a shell or shell-like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax.

If you wish to define a function-like macro on the command line, write its argument list with surrounding parentheses before the equals sign (if any). Parentheses are meaningful to most shells, so you will need to quote the option. With sh and csh, `−D'name(args...)=definition'` works.

−D and **−U** options are processed in the order they are given on the command line. All −imacros file and −include file options are processed after all −D and −U options.

−H Print the name of each header file used, in addition to other normal activities. Each name is indented to show how deep in the `'#include'` stack it is.

−P Inhibit generation of linemarkers in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code, and will be sent to a program which might be confused by the linemarkers.

−U*name*
Cancel any previous definition of *name*, either built in or provided with a **−D** option.

Options to request or suppress errors and warnings

Errors are diagnostic messages that report that the GNU Fortran compiler cannot compile the relevant piece of source code. The compiler will continue to process the program in an attempt to report further errors to aid in debugging, but will not produce any compiled output.

Warnings are diagnostic messages that report constructions which are not inherently erroneous but which are risky or suggest there is likely to be a bug in the program. Unless **−Werror** is specified, they do not prevent compilation of the program.

You can request many specific warnings with options beginning **−W**, for example **−Wimplicit** to request warnings on implicit declarations. Each of these specific warning options also has a negative form beginning **−Wno−** to turn off warnings; for example, **−Wno−implicit**. This manual lists only one of the two forms, whichever is not the default.

These options control the amount and kinds of errors and warnings produced by GNU Fortran:

−fmax−errors=*n*
Limits the maximum number of error messages to *n*, at which point GNU Fortran bails out rather than attempting to continue processing the source code. If *n* is 0, there is no limit on the number of error messages produced.

−fsyntax−only
Check the code for syntax errors, but do not actually compile it. This will generate module files for each module present in the code, but no other output file.

−Wpedantic
−pedantic
Issue warnings for uses of extensions to Fortran. **−pedantic** also applies to C−language constructs where they occur in GNU Fortran source files, such as use of **\e** in a character constant within a directive like `#include`.

Valid Fortran programs should compile properly with or without this option. However, without this option, certain GNU extensions and traditional Fortran features are supported as well. With this

option, many of them are rejected.

Some users try to use **−pedantic** to check programs for conformance. They soon find that it does not do quite what they want−−−it finds some nonstandard practices, but not all. However, improvements to GNU Fortran in this area are welcome.

This should be used in conjunction with **−std=f95**, **−std=f2003**, **−std=f2008** or **−std=f2018**.

−pedantic−errors
Like **−pedantic**, except that errors are produced rather than warnings.

−Wall
Enables commonly used warning options pertaining to usage that we recommend avoiding and that we believe are easy to avoid. This currently includes**−W aliasing**, **−Wampersand**, **−Wconversion**, **−Wsurprising**, **−Wc−binding−type**, **−Wintrinsics−std**, **−Wtabs**, **−Wintrinsic−shadow**, **−Wline−truncation**, **−Wtarget−lifetime**, **−Winteger−division**, **−Wreal−q−constant**, **−Wunused** and **−Wundefined−do−loop**.

−Waliasing
Warn about possible aliasing of dummy arguments. Specifically, it warns if the same actual argument is associated with a dummy argument with `INTENT(IN)` and a dummy argument with `INTENT(OUT)` in a call with an explicit interface.

The following example will trigger the warning.

```
interface
  subroutine bar(a,b)
    integer, intent(in) :: a
    integer, intent(out) :: b
  end subroutine
end interface
integer :: a

call bar(a,a)
```

−Wampersand
Warn about missing ampersand in continued character constants. The warning is given with **−Wampersand**, **−pedantic**, **−std=f95**, **−std=f2003**, **−std=f2008** and **−std=f2018**. Note: With no ampersand given in a continued character constant, GNU Fortran assumes continuation at the first non-comment, non-whitespace character after the ampersand that initiated the continuation.

−Warray−temporaries
Warn about array temporaries generated by the compiler. The information generated by this warning is sometimes useful in optimization, in order to avoid such temporaries.

−Wc−binding−type
Warn if the a variable might not be C interoperable. In particular, warn if the variable has been declared using an intrinsic type with default kind instead of using a kind parameter defined for C interoperability in the intrinsic `ISO_C_Binding` module. This option is implied by **−Wall**.

−Wcharacter−truncation
Warn when a character assignment will truncate the assigned string.

−Wline−truncation
Warn when a source code line will be truncated. This option is implied by **−Wall**. For free-form source code, the default is **−Werror=line−truncation** such that truncations are reported as error.

−Wconversion
Warn about implicit conversions that are likely to change the value of the expression after conversion. Implied by **−Wall**.

−Wconversion−extra
Warn about implicit conversions between different types and kinds. This option does *not* imply **−Wconversion**.

−Wextra
Enables some warning options for usages of language features which may be problematic. This currently includes **−Wcompare−reals**, **−Wunused−parameter** and **−Wdo−subscript**.

−Wfrontend−loop−interchange
Warn when using **−ffrontend−loop−interchange** for performing loop interchanges.

−Wimplicit−interface
Warn if a procedure is called without an explicit interface. Note this only checks that an explicit interface is present. It does not check that the declared interfaces are consistent across program units.

−Wimplicit−procedure
Warn if a procedure is called that has neither an explicit interface nor has been declared as EXTERNAL.

−Winteger−division
Warn if a constant integer division truncates its result. As an example, 3/5 evaluates to 0.

−Wintrinsics−std
Warn if **gfortran** finds a procedure named like an intrinsic not available in the currently selected standard (with **−std**) and treats it as EXTERNAL procedure because of this. **−fall−intrinsics** can be used to never trigger this behavior and always link to the intrinsic regardless of the selected standard.

−Wno−overwrite−recursive
Do not warn when **−fno−automatic** is used with **−frecursive**. Recursion will be broken if the relevant local variables do not have the attribute AUTOMATIC explicitly declared. This option can be used to suppress the warning when it is known that recursion is not broken. Useful for build environments that use **−Werror**.

−Wreal−q−constant
Produce a warning if a real-literal-constant contains a q exponent-letter.

−Wsurprising
Produce a warning when ‘‘suspicious’’ code constructs are encountered. While technically legal these usually indicate that an error has been made.

This currently produces a warning under the following circumstances:

* An INTEGER SELECT construct has a CASE that can never be matched as its lower value is greater than its upper value.
* A LOGICAL SELECT construct has three CASE statements.
* A TRANSFER specifies a source that is shorter than the destination.
* The type of a function result is declared more than once with the same type. If **−pedantic** or standard-conforming mode is enabled, this is an error.
* A CHARACTER variable is declared with negative length.

−Wtabs
By default, tabs are accepted as whitespace, but tabs are not members of the Fortran Character Set. For continuation lines, a tab followed by a digit between 1 and 9 is supported. **−Wtabs** will cause a warning to be issued if a tab is encountered. Note, **−Wtabs** is active for **−pedantic**, **−std=f95**, **−std=f2003**, **−std=f2008**, **−std=f2018** and **−Wall**.

−Wundefined−do−loop
Warn if a DO loop with step either 1 or −1 yields an underflow or an overflow during iteration of an induction variable of the loop. This option is implied by **−Wall**.

−Wunderflow
Produce a warning when numerical constant expressions are encountered, which yield an UNDERFLOW during compilation. Enabled by default.

−Wintrinsic−shadow
Warn if a user-defined procedure or module procedure has the same name as an intrinsic; in this case, an explicit interface or `EXTERNAL` or `INTRINSIC` declaration might be needed to get calls later resolved to the desired intrinsic/procedure. This option is implied by **−Wall**.

−Wuse−without−only
Warn if a `USE` statement has no `ONLY` qualifier and thus implicitly imports all public entities of the used module.

−Wunused−dummy−argument
Warn about unused dummy arguments. This option is implied by **−Wall**.

−Wunused−parameter
Contrary to **gcc**'s meaning of **−Wunused−parameter**, **gfortran**'s implementation of this option does not warn about unused dummy arguments (see **−Wunused−dummy−argument**), but about unused `PARAMETER` values. **−Wunused−parameter** is implied by **−Wextra** if also **−Wunused** or **−Wall** is used.

−Walign−commons
By default, **gfortran** warns about any occasion of variables being padded for proper alignment inside a `COMMON` block. This warning can be turned off via **−Wno−align−commons**. See also **−falign−commons**.

−Wfunction−elimination
Warn if any calls to impure functions are eliminated by the optimizations enabled by the **−ffrontend−optimize** option. This option is implied by **−Wextra**.

−Wrealloc−lhs
Warn when the compiler might insert code to for allocation or reallocation of an allocatable array variable of intrinsic type in intrinsic assignments. In hot loops, the Fortran 2003 reallocation feature may reduce the performance. If the array is already allocated with the correct shape, consider using a whole-array array-spec (e.g. `(:,:,:)`) for the variable on the left-hand side to prevent the reallocation check. Note that in some cases the warning is shown, even if the compiler will optimize reallocation checks away. For instance, when the right-hand side contains the same variable multiplied by a scalar. See also**−fr ealloc−lhs**.

−Wrealloc−lhs−all
Warn when the compiler inserts code to for allocation or reallocation of an allocatable variable; this includes scalars and derived types.

−Wcompare−reals
Warn when comparing real or complex types for equality or inequality. This option is implied by **−Wextra**.

−Wtarget−lifetime
Warn if the pointer in a pointer assignment might be longer than the its target. This option is implied by **−Wall**.

−Wzerotrip
Warn if a `DO` loop is known to execute zero times at compile time. This option is implied by **−Wall**.

−Wdo−subscript
Warn if an array subscript inside a DO loop could lead to an out-of-bounds access even if the compiler cannot prove that the statement is actually executed, in cases like

```
real a(3)
do i=1,4
  if (condition(i)) then
    a(i) = 1.2
  end if
end do
```

This option is implied by **−Wextra**.

−Werror
Turns all warnings into errors.

Some of these have no effect when compiling programs written in Fortran.

Options for debugging your program or GNU Fortran
GNU Fortran has various special options that are used for debugging either your program or the GNU Fortran compiler.

−fdump−fortran−original
Output the internal parse tree after translating the source program into internal representation. This option is mostly useful for debugging the GNU Fortran compiler itself. The output generated by this option might change between releases. This option may also generate internal compiler errors for features which have only recently been added.

−fdump−fortran−optimized
Output the parse tree after front-end optimization. Mostly useful for debugging the GNU Fortran compiler itself. The output generated by this option might change between releases. This option may also generate internal compiler errors for features which have only recently been added.

−fdump−parse−tree
Output the internal parse tree after translating the source program into internal representation. Mostly useful for debugging the GNU Fortran compiler itself. The output generated by this option might change between releases. This option may also generate internal compiler errors for features which have only recently been added. This option is deprecated; use `−fdump−fortran−original` instead.

−fdebug−aux−vars
Renames internal variables created by the gfortran front end and makes them accessible to a debugger. The name of the internal variables then start with upper-case letters followed by an underscore. This option is useful for debugging the compiler's code generation together with `−fdump−tree−original` and enabling debugging of the executable program by using `−g` or `−ggdb3`.

−fdump−fortran−global
Output a list of the global identifiers after translating into middle-end representation. Mostly useful for debugging the GNU Fortran compiler itself. The output generated by this option might change between releases. This option may also generate internal compiler errors for features which have only recently been added.

−ffpe−trap=*list*
Specify a list of floating point exception traps to enable. On most systems, if a floating point exception occurs and the trap for that exception is enabled, a SIGFPE signal will be sent and the program being aborted, producing a core file useful for debugging. *list* is a (possibly empty) comma-separated list of the following exceptions: **invalid** (invalid floating point operation, such as `SQRT(−1.0)`), **zero** (division by zero), **overflow** (overflow in a floating point operation), **underflow** (underflow in a floating point operation), **inexact** (loss of precision during operation), and **denormal** (operation performed on a denormal value). The first five exceptions correspond to the five IEEE 754 exceptions, whereas the last one (**denormal**) is not part of the IEEE 754 standard but is available on some common architectures such as x86.

The first three exceptions (**invalid**, **zero**, and **overflow**) often indicate serious errors, and unless the

program has provisions for dealing with these exceptions, enabling traps for these three exceptions is probably a good idea.

If the option is used more than once in the command line, the lists will be joined: '`ffpe-trap=`*list1* `ffpe-trap=`*list2*' is equivalent to `ffpe-trap=`*list1*,*list2*.

Note that once enabled an exception cannot be disabled (no negative form).

Many, if not most, floating point operations incur loss of precision due to rounding, and hence the `ffpe-trap=inexact` is likely to be uninteresting in practice.

By default no exception traps are enabled.

-ffpe-summary=*list*

Specify a list of floating-point exceptions, whose flag status is printed to `ERROR_UNIT` when invoking `STOP` and `ERROR STOP`. *list* can be either **none**, **all** or a comma-separated list of the following exceptions: **invalid**, **zero**, **overflow**, **underflow**, **inexact** and **denormal**. (See **-ffpe-trap** for a description of the exceptions.)

If the option is used more than once in the command line, only the last one will be used.

By default, a summary for all exceptions but **inexact** is shown.

-fno-backtrace

When a serious runtime error is encountered or a deadly signal is emitted (segmentation fault, illegal instruction, bus error, floating-point exception, and the other POSIX signals that have the action **core**), the Fortran runtime library tries to output a backtrace of the error. `-fno-backtrace` disables the backtrace generation. This option only has influence for compilation of the Fortran main program.

Options for directory search

These options affect how GNU Fortran searches for files specified by the `INCLUDE` directive and where it searches for previously compiled modules.

It also affects the search paths used by **cpp** when used to preprocess Fortran source.

-I*dir*

These affect interpretation of the `INCLUDE` directive (as well as of the `#include` directive of the **cpp** preprocessor).

Also note that the general behavior of **-I** and `INCLUDE` is pretty much the same as of **-I** with `#include` in the **cpp** preprocessor, with regard to looking for *header.gcc* files and other such things.

This path is also used to search for *.mod* files when previously compiled modules are required by a `USE` statement.

-J*dir*

This option specifies where to put *.mod* files for compiled modules. It is also added to the list of directories to searched by an `USE` statement.

The default is the current directory.

-fintrinsic-modules-path *dir*

This option specifies the location of pre-compiled intrinsic modules, if they are not in the default location expected by the compiler.

Influencing the linking step

These options come into play when the compiler links object files into an executable output file. They are meaningless if the compiler is not doing a link step.

-static-libgfortran

On systems that provide *libgfortran* as a shared and a static library, this option forces the use of the static version. If no shared version of *libgfortran* was built when the compiler was configured, this option has no effect.

Influencing runtime behavior

These options affect the runtime behavior of programs compiled with GNU Fortran.

−fconvert=*conversion*

Specify the representation of data for unformatted files. Valid values for conversion are: **native**, the default; **swap**, swap between big− and little-endian; **big-endian**, use big-endian representation for unformatted files; **little-endian**, use little-endian representation for unformatted files.

This option has an effect only when used in the main program. The `CONVERT` *specifier and the GFORTRAN_CONVERT_UNIT environment variable override the default specified by* ***−fconvert****.*

−frecord−marker=*length*

Specify the length of record markers for unformatted files. Valid values for *length* are 4 and 8. Default is 4. *This is different from previous versions of* **gfortran**, which specified a default record marker length of 8 on most systems. If you want to read or write files compatible with earlier versions of **gfortran**, use **−frecord−marker=8**.

−fmax−subrecord−length=*length*

Specify the maximum length for a subrecord. The maximum permitted value for length is 2147483639, which is also the default. Only really useful for use by the gfortran testsuite.

−fsign−zero

When enabled, floating point numbers of value zero with the sign bit set are written as negative number in formatted output and treated as negative in the `SIGN` intrinsic. **−fno−sign−zero** does not print the negative sign of zero values (or values rounded to zero for I/O) and regards zero as positive number in the `SIGN` intrinsic for compatibility with Fortran 77. The default is **−fsign−zero**.

Options for code generation conventions

These machine-independent options control the interface conventions used in code generation.

Most of them have both positive and negative forms; the negative form of **−ffoo** would be **−fno−foo**. In the table below, only one of the forms is listed−−−the one which is not the default. You can figure out the other form by either removing **no−** or adding it.

−fno−automatic

Treat each program unit (except those marked as RECURSIVE) as if the `SAVE` statement were specified for every local variable and array referenced in it. Does not affect common blocks. (Some Fortran compilers provide this option under the name **−static** or **−save**.) The default, which is **−fautomatic**, uses the stack for local variables smaller than the value given by **−fmax−stack−var−size**. Use the option **−frecursive** to use no static memory.

Local variables or arrays having an explicit `SAVE` attribute are silently ignored unless the **−pedantic** option is added.

−ff2c

Generate code designed to be compatible with code generated by **g77** and **f2c**.

The calling conventions used by **g77** (originally implemented in **f2c**) require functions that return type default `REAL` to actually return the C type `double`, and functions that return type `COMPLEX` to return the values via an extra argument in the calling sequence that points to where to store the return value. Under the default GNU calling conventions, such functions simply return their results as they would in GNU C−−−default `REAL` functions return the C type `float`, and `COMPLEX` functions return the GNU C type `complex`. Additionally, this option implies the **−fsecond−underscore** option, unless **−fno−second−underscore** is explicitly requested.

This does not affect the generation of code that interfaces with the **libgfortran** library.

Caution: It is not a good idea to mix Fortran code compiled with **−ff2c** with code compiled with the default **−fno−f2c** calling conventions as, calling `COMPLEX` or default `REAL` functions between program parts which were compiled with different calling conventions will break at execution time.

Caution: This will break code which passes intrinsic functions of type default `REAL` or `COMPLEX` as

actual arguments, as the library implementations use the **−fno−f2c** calling conventions.

−fno−underscoring
Do not transform names of entities specified in the Fortran source file by appending underscores to them.

With **−funderscoring** in effect, GNU Fortran appends one underscore to external names with no underscores. This is done to ensure compatibility with code produced by many UNIX Fortran compilers.

Caution: The default behavior of GNU Fortran is incompatible with **f2c** and **g77**, please use the **−ff2c** option if you want object files compiled with GNU Fortran to be compatible with object code created with these tools.

Use of **−fno−underscoring** is not recommended unless you are experimenting with issues such as integration of GNU Fortran into existing system environments (vis−à−vis existing libraries, tools, and so on).

For example, with **−funderscoring**, and assuming that `j()` and `max_count()` are external functions while `my_var` and `lvar` are local variables, a statement like

```
I = J() + MAX_COUNT (MY_VAR, LVAR)
```

is implemented as something akin to:

```
i = j_() + max_count__(&my_var__, &lvar);
```

With **−fno−underscoring**, the same statement is implemented as:

```
i = j() + max_count(&my_var, &lvar);
```

Use of **−fno−underscoring** allows direct specification of user-defined names while debugging and when interfacing GNU Fortran code with other languages.

Note that just because the names match does *not* mean that the interface implemented by GNU Fortran for an external name matches the interface implemented by some other language for that same name. That is, getting code produced by GNU Fortran to link to code produced by some other compiler using this or any other method can be only a small part of the overall solution−−−getting the code generated by both compilers to agree on issues other than naming can require significant effort, and, unlike naming disagreements, linkers normally cannot detect disagreements in these other areas.

Also, note that with **−fno−underscoring**, the lack of appended underscores introduces the very real possibility that a user-defined external name will conflict with a name in a system library, which could make finding unresolved-reference bugs quite difficult in some cases−−−they might occur at program run time, and show up only as buggy behavior at run time.

In future versions of GNU Fortran we hope to improve naming and linking issues so that debugging always involves using the names as they appear in the source, even if the names as seen by the linker are mangled to prevent accidental linking between procedures with incompatible interfaces.

−fsecond−underscore
By default, GNU Fortran appends an underscore to external names. If this option is used GNU Fortran appends two underscores to names with underscores and one underscore to external names with no underscores. GNU Fortran also appends two underscores to internal names with underscores to avoid naming collisions with external names.

This option has no effect if **−fno−underscoring** is in effect. It is implied by the**−ff2c** option.

Otherwise, with this option, an external name such as `MAX_COUNT` is implemented as a reference to the link-time external symbol `max_count__`, instead of `max_count_`. This is required for compatibility with **g77** and **f2c**, and is implied by use of the **−ff2c** option.

−fcoarray=<*keyword*>

none
Disable coarray support; using coarray declarations and image-control statements will produce a compile-time error. (Default)

single
Single-image mode, i.e. `num_images()` is always one.

lib Library-based coarray parallelization; a suitable GNU Fortran coarray library needs to be linked.

−fcheck=<*keyword*>
Enable the generation of run-time checks; the argument shall be a comma-delimited list of the following keywords. Prefixing a check with**no−** disables it if it w as activated by a previous specification.

all Enable all run-time test of **−fcheck**.

array-temps
Warns at run time when for passing an actual argument a temporary array had to be generated. The information generated by this warning is sometimes useful in optimization, in order to avoid such temporaries.

Note: The warning is only printed once per location.

bits
Enable generation of run-time checks for invalid arguments to the bit manipulation intrinsics.

bounds
Enable generation of run-time checks for array subscripts and against the declared minimum and maximum values. It also checks array indices for assumed and deferred shape arrays against the actual allocated bounds and ensures that all string lengths are equal for character array constructors without an explicit typespec.

Some checks require that **−fcheck=bounds** is set for the compilation of the main program.

Note: In the future this may also include other forms of checking, e.g., checking substring references.

do Enable generation of run-time checks for invalid modification of loop iteration variables.

mem
Enable generation of run-time checks for memory allocation. Note: This option does not affect explicit allocations using the `ALLOCATE` statement, which will be always checked.

pointer
Enable generation of run-time checks for pointers and allocatables.

recursion
Enable generation of run-time checks for recursively called subroutines and functions which are not marked as recursive. See also **−frecursive**. Note: This check does not work for OpenMP programs and is disabled if used together with **−frecursive** and **−fopenmp**.

Example: Assuming you have a file *foo.f90*, the command

```
gfortran −fcheck=all,no−array−temps foo.f90
```

will compile the file with all checks enabled as specified above except warnings for generated array temporaries.

−fbounds−check
Deprecated alias for **−fcheck=bounds**.

−ftail−call−workaround

−ftail−call−workaround=*n*
Some C interfaces to Fortran codes violate the gfortran ABI by omitting the hidden character length arguments as described in
This can lead to crashes because pushing arguments for tail calls can overflow the stack.

To provide a workaround for existing binary packages, this option disables tail call optimization for gfortran procedures with character arguments. With **−ftail−call−workaround=2** tail call optimization is disabled in all gfortran procedures with character arguments, with **−ftail−call−workaround=1** or equivalent **−ftail−call−workaround** only in gfortran procedures with character arguments that call implicitly prototyped procedures.

Using this option can lead to problems including crashes due to insufficient stack space.

It is *very strongly* recommended to fix the code in question. The **−fc−prototypes−external** option can be used to generate prototypes which conform to gfortran's ABI, for inclusion in the source code.

Support for this option will likely be withdrawn in a future release of gfortran.

The negative form, **−fno−tail−call−workaround** or equivalent **−ftail−call−workaround=0**, can be used to disable this option.

Default is currently **−ftail−call−workaround**, this will change in future releases.

−fcheck−array−temporaries
Deprecated alias for **−fcheck=array−temps**.

−fmax−array−constructor=*n*
This option can be used to increase the upper limit permitted in array constructors. The code below requires this option to expand the array at compile time.

```
program test
implicit none
integer j
integer, parameter :: n = 100000
integer, parameter :: i(n) = (/ (2*j, j = 1, n) /)
print '(10(I0,1X))', i
end program test
```

Caution: This option can lead to long compile times and excessively large object files.

The default value for *n* is 65535.

−fmax−stack−var−size=*n*
This option specifies the size in bytes of the largest array that will be put on the stack; if the size is exceeded static memory is used (except in procedures marked as RECURSIVE). Use the option **−frecursive** to allow for recursive procedures which do not have a RECURSIVE attribute or for parallel programs. Use **−fno−automatic** to never use the stack.

This option currently only affects local arrays declared with constant bounds, and may not apply to all character variables. Future versions of GNU Fortran may improve this behavior.

The default value for *n* is 65536.

−fstack−arrays
Adding this option will make the Fortran compiler put all arrays of unknown size and array temporaries onto stack memory. If your program uses very large local arrays it is possible that you will have to extend your runtime limits for stack memory on some operating systems. This flag is enabled by default at optimization level **−Ofast** unless **−fmax−stack−var−size** is specified.

−fpack−derived
This option tells GNU Fortran to pack derived type members as closely as possible. Code compiled with this option is likely to be incompatible with code compiled without this option, and may execute slower.

−frepack−arrays
In some circumstances GNU Fortran may pass assumed shape array sections via a descriptor describing a noncontiguous area of memory. This option adds code to the function prologue to repack the data into a contiguous block at runtime.

This should result in faster accesses to the array. However it can introduce significant overhead to the function call, especially when the passed data is noncontiguous.

−fshort−enums
This option is provided for interoperability with C code that was compiled with the **−fshort−enums** option. It will make GNU Fortran choose the smallest `INTEGER` kind a given enumerator set will fit in, and give all its enumerators this kind.

−finline−arg−packing
When passing an assumed-shape argument of a procedure as actual argument to an assumed-size or explicit size or as argument to a procedure that does not have an explicit interface, the argument may have to be packed, that is put into contiguous memory. An example is the call to `foo` in

```
subroutine foo(a)
   real, dimension(*) :: a
end subroutine foo
subroutine bar(b)
   real, dimension(:) :: b
   call foo(b)
end subroutine bar
```

When **−finline−arg−packing** is in effect, this packing will be performed by inline code. This allows for more optimization while increasing code size.

−finline−arg−packing is implied by any of the **−O** options except when optimizing for size via **−Os**. If the code contains a very large number of argument that have to be packed, code size and also compilation time may become excessive. If that is the case, it may be better to disable this option. Instances of packing can be found by using by using **−Warray−temporaries**.

−fexternal−blas
This option will make **gfortran** generate calls to BLAS functions for some matrix operations like `MATMUL`, instead of using our own algorithms, if the size of the matrices involved is larger than a given limit (see **−fblas−matmul−limit**). This may be profitable if an optimized vendor BLAS library is available. The BLAS library will have to be specified at link time.

−fblas−matmul−limit=*n*
Only significant when **−fexternal−blas** is in effect. Matrix multiplication of matrices with size larger than (or equal to) *n* will be performed by calls to BLAS functions, while others will be handled by **gfortran** internal algorithms. If the matrices involved are not square, the size comparison is performed using the geometric mean of the dimensions of the argument and result matrices.

The default value for *n* is 30.

−finline−matmul−limit=*n*
When front-end optimization is active, some calls to the `MATMUL` intrinsic function will be inlined. This may result in code size increase if the size of the matrix cannot be determined at compile time, as code for both cases is generated. Setting `−finline−matmul−limit=0` will disable inlining in all cases. Setting this option with a value of *n* will produce inline code for matrices with size up to *n*. If the matrices involved are not square, the size comparison is performed using the geometric mean of the dimensions of the argument and result matrices.

The default value for *n* is 30. The `−fblas−matmul−limit` can be used to change this value.

−frecursive
Allow indirect recursion by forcing all local arrays to be allocated on the stack. This flag cannot be used together with **−fmax−stack−var−size=** or **−fno−automatic**.

−finit−local−zero
−finit−derived
−finit−integer=*n*
−finit−real=<*zero*|*inf*|*−inf*|*nan*|*snan*>
−finit−logical=<*true*|*false*>
−finit−character=*n*

The **−finit−local−zero** option instructs the compiler to initialize local `INTEGER`, `REAL`, and `COMPLEX` variables to zero, `LOGICAL` variables to false, and `CHARACTER` variables to a string of null bytes. Finer-grained initialization options are provided by the **−finit−integer=***n*, **−finit−real=**<*zero*|*inf*|*−inf*|*nan*|*snan*> (which also initializes the real and imaginary parts of local `COMPLEX` variables), **−finit−logical=**<*true*|*false*>, and **−finit−character=***n* (where *n* is an ASCII character value) options.

With **−finit−derived**, components of derived type variables will be initialized according to these flags. Components whose type is not covered by an explicit **−finit−*** flag will be treated as described above with **−finit−local−zero**.

These options do not initialize

* objects with the POINTER attribute
* allocatable arrays
* variables that appear in an `EQUIVALENCE` statement.

(These limitations may be removed in future releases).

Note that the **−finit−real=nan** option initializes `REAL` and `COMPLEX` variables with a quiet NaN. For a signalling NaN use **−finit−real=snan**; note, however, that compile-time optimizations may convert them into quiet NaN and that trapping needs to be enabled (e.g. via **−ffpe−trap**).

The **−finit−integer** option will parse the value into an integer of type `INTEGER(kind=C_LONG)` on the host. Said value is then assigned to the integer variables in the Fortran code, which might result in wraparound if the value is too large for the kind.

Finally, note that enabling any of the **−finit−*** options will silence warnings that would have been emitted by **−Wuninitialized** for the affected local variables.

−falign−commons

By default, **gfortran** enforces proper alignment of all variables in a `COMMON` block by padding them as needed. On certain platforms this is mandatory, on others it increases performance. If a `COMMON` block is not declared with consistent data types everywhere, this padding can cause trouble, and **−fno−align−commons** can be used to disable automatic alignment. The same form of this option should be used for all files that share a `COMMON` block. To avoid potential alignment issues in `COMMON` blocks, it is recommended to order objects from largest to smallest.

−fno−protect−parens

By default the parentheses in expression are honored for all optimization levels such that the compiler does not do any re-association. Using **−fno−protect−parens** allows the compiler to reorder `REAL` and `COMPLEX` expressions to produce faster code. Note that for the re-association optimization **−fno−signed−zeros** and **−fno−trapping−math** need to be in effect. The parentheses protection is enabled by default, unless **−Ofast** is given.

−frealloc−lhs

An allocatable left-hand side of an intrinsic assignment is automatically (re)allocated if it is either unallocated or has a different shape. The option is enabled by default except when **−std=f95** is given. See also **−Wrealloc−lhs**.

−faggressive−function−elimination

Functions with identical argument lists are eliminated within statements, regardless of whether these functions are marked `PURE` or not. For example, in

```
a = f(b,c) + f(b,c)
```

there will only be a single call to `f`. This option only works if **−ffrontend−optimize** is in effect.

−ffrontend−optimize
This option performs front-end optimization, based on manipulating parts the Fortran parse tree. Enabled by default by any **−O** option except **−O0** and **−Og**. Optimizations enabled by this option include:

*<inlining calls to `MATMUL`,>
*<elimination of identical function calls within expressions,>
*<removing unnecessary calls to `TRIM` in comparisons and assignments,>
*<replacing `TRIM(a)` with `a(1:LEN_TRIM(a))` and>
*<short−circuiting of logical operators (`.AND.` and `.OR.`).>

It can be deselected by specifying **−fno−frontend−optimize**.

−ffrontend−loop−interchange
Attempt to interchange loops in the Fortran front end where profitable. Enabled by default by any **−O** option. At the moment, this option only affects `FORALL` and `DO CONCURRENT` statements with several forall triplets.

ENVIRONMENT

The **gfortran** compiler currently does not make use of any environment variables to control its operation above and beyond those that affect the operation of **gcc**.

BUGS

For instructions on reporting bugs, see <**file:///usr/share/doc/gcc−11/README.Bugs**>.

SEE ALSO

gpl (7), **gfdl** (7), **fsf−funding** (7), **cpp** (1), **gcov** (1), **gcc** (1), **as** (1), **ld** (1), **gdb** (1), **dbx** (1) and the Info entries for *gcc*, *cpp*, *gfortran*, *as*, *ld*, *binutils* and *gdb*.

AUTHOR

See the Info entry for **gfortran** for contributors to GCC and GNU Fortran.

COPYRIGHT