The GNU D Compiler

For GCC version 17.0.0 (pre-release)

(GCC)

David Friedman, Iain Buclaw

Published by the Free Software Foundation
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301, USA

Table of Contents

1 Invoking gdc

The `gdc` command is the GNU compiler for the D language and supports many of the same options as `gcc`. See Section "Option Summary" in *Using the GNU Compiler Collection (GCC)*. This manual only documents the options specific to `gdc`.

1.1 Input and Output files

For any given input file, the file name suffix determines what kind of compilation is done. The following kinds of input file names are supported:

file`.d` D source files.

file`.dd` Ddoc source files.

file`.di` D interface files.

You can specify more than one input file on the `gdc` command line, each being compiled separately in the compilation process. If you specify a `-o` *file* option, all the input files are compiled together, producing a single output file, named *file*. This is allowed even when using `-S` or `-c`.

A D interface file contains only what an import of the module needs, rather than the whole implementation of that module. They can be created by `gdc` from a D source file by using the `-H` option. When the compiler resolves an import declaration, it searches for matching `.di` files first, then for `.d`.

A Ddoc source file contains code in the D macro processor language. It is primarily designed for use in producing user documentation from embedded comments, with a slight affinity towards HTML generation. If a `.d` source file starts with the string `Ddoc` then it is treated as general purpose documentation, not as a D source file.

1.2 Runtime Options

These options affect the runtime behavior of programs compiled with `gdc`.

`-fall-instantiations`
: Generate code for all template instantiations. The default template emission strategy is to not generate code for declarations that were either instantiated speculatively, such as from `__traits(compiles, ...)`, or that come from an imported module not being compiled.

`-fno-assert`
: Turn off code generation for `assert` contracts.

`-fno-bounds-check`
: Turns off array bounds checking for all functions, which can improve performance for code that uses arrays extensively. Note that this can result in unpredictable behavior if the code in question actually does violate array bounds constraints. It is safe to use this option if you are sure that your code never throws a `RangeError`.

`-fbounds-check=`*value*
: An alternative to `-fbounds-check` that allows more control as to where bounds checking is turned on or off. The following values are supported:

  ‘`on`’ Turns on array bounds checking for all functions.

  ‘`safeonly`’
  Turns on array bounds checking only for `@safe` functions.

  ‘`off`’ Turns off array bounds checking completely.

`-fno-builtin`
: Don’t recognize built-in functions unless they begin with the prefix ‘`__builtin_`’. By default, the compiler will recognize when a function in the `core.stdc` package is a built-in function.

`-fcheckaction=`*value*
: This option controls what code is generated on an assertion, bounds check, or final switch failure. The following values are supported:

  ‘`context`’ Throw an `AssertError` with extra context information.

  ‘`halt`’ Halt the program execution.

  ‘`throw`’ Throw an `AssertError` (the default).

`-fdebug`

`-fdebug=`*value*
: Turn on compilation of conditional `debug` code into the program. The `-fdebug` option itself sets the debug level to `1`, while `-fdebug=` enables `debug` code that are identified by any of the following values:

  ‘`ident`’ Turns on compilation of any `debug` code identified by *ident*.

`-fno-druntime`
: Implements `https://dlang.org/spec/betterc.html`. Assumes that compilation targets an environment without a D runtime library.

  This is equivalent to compiling with the following options:

  ```
  gdc -nophoboslib -fno-exceptions -fno-moduleinfo -fno-rtti
  ```

`-fextern-std=`*standard*
: Sets the C++ name mangling compatibility to the version identified by *standard*. The following values are supported:

  ‘`c++98`’

  ‘`c++03`’ Sets `__traits(getTargetInfo, "cppStd")` to `199711`.

  ‘`c++11`’ Sets `__traits(getTargetInfo, "cppStd")` to `201103`.

  ‘`c++14`’ Sets `__traits(getTargetInfo, "cppStd")` to `201402`.

  ‘`c++17`’ Sets `__traits(getTargetInfo, "cppStd")` to `201703`. This is the default.

  ‘`c++20`’ Sets `__traits(getTargetInfo, "cppStd")` to `202002`.

'c++23' Sets `__traits(getTargetInfo, "cppStd")` to 202302.

`-finclude-imports`
: Include imported modules in the compilation, as if they were given on the command line. When this option is enabled, all imported modules are compiled except those that are part of libphobos.

`-fno-invariants`
: Turns off code generation for class `invariant` contracts.

`-fmain`
: Generates a default `main()` function when compiling. This is useful when unittesting a library, as it enables running the unittests in a library without having to manually define an entry-point function. This option does nothing when `main` is already defined in user code.

`-fno-moduleinfo`
: Turns off generation of the `ModuleInfo` and related functions that would become unreferenced without it, which may allow linking to programs not written in D. Functions that are not be generated include module constructors and destructors (`static this` and `static ~this`), `unittest` code, and `DSO` registry functions for dynamically linked code.

`-fonly=`*`filename`*
: Tells the compiler to parse and run semantic analysis on all modules on the command line, but only generate code for the module specified by *filename*.

`-fno-postconditions`
: Turns off code generation for postcondition `out` contracts.

`-fno-preconditions`
: Turns off code generation for precondition `in` contracts.

`-fpreview=`*`id`*
: Turns on an upcoming D language change identified by *id*. The following values are supported:

  'all' Turns on all upcoming D language features.

  'bitfields'
  Implements bit-fields in D.

  'dip1000' Implements `https://github.com/dlang/DIPs/blob/master/DIPs/other/DIP1000.md` (Scoped pointers).

  'dip1008' Implements `https://github.com/dlang/DIPs/blob/master/DIPs/other/DIP1008.md` (Allow exceptions in `@nogc` code).

  'dip1021' Implements `https://github.com/dlang/DIPs/blob/master/DIPs/accepted/DIP1021.md` (Mutable function arguments).

  'dip25' Implements `https://github.com/dlang/DIPs/blob/master/DIPs/archive/DIP25.md` (Sealed references).

  'dtorfields'
  Turns on generation for destructing fields of partially constructed objects.

'`fieldwise`'
: Turns on generation of struct equality to use field-wise comparisons.

'`fixaliasthis`'
: Implements new lookup rules that check the current scope for `alias this` before searching in upper scopes.

'`fiximmutableconv`'
: Disallows unsound immutable conversions that were formerly incorrectly permitted.

'`in`'
: Implements `in` parameters to mean `scope const [ref]` and accepts rvalues.

'`inclusiveincontracts`'
: Implements `in` contracts of overridden methods to be a superset of parent contract.

'`nosharedaccess`'
: Turns off and disallows all access to shared memory objects.

'`rvaluerefparam`'
: Implements rvalue arguments to `ref` parameters.

'`systemvariables`'
: Disables access to variables marked `@system` from `@safe` code.

`-frelease`

Turns on compiling in release mode, which means not emitting runtime checks for contracts and asserts. Array bounds checking is not done for `@system` and `@trusted` functions, and assertion failures are undefined behavior.

This is equivalent to compiling with the following options:

```
gdc -fno-assert -fbounds-check=safe -fno-invariants \
    -fno-postconditions -fno-preconditions -fno-switch-errors
```

`-frevert=`

Turns off a D language feature identified by *id*. The following values are supported:

'`all`'
: Turns off all revertable D language features.

'`dip1000`'
: Reverts `https://github.com/dlang/DIPs/blob/master/DIPs/other/DIP1000.md` (Scoped pointers).

'`dip25`'
: Reverts `https://github.com/dlang/DIPs/blob/master/DIPs/archive/DIP25.md` (Sealed references).

'`dtorfields`'
: Turns off generation for destructing fields of partially constructed objects.

'`intpromote`'
: Turns off C-style integral promotion for unary `+`, `-` and `~` expressions.

`-fno-rtti`
: Turns off generation of run-time type information for all user defined types. Any code that uses features of the language that require access to this information will result in an error.

`-fno-switch-errors`
: This option controls what code is generated when no case is matched in a `final switch` statement. The default run time behavior is to throw a `SwitchError`. Turning off `-fswitch-errors` means that instead the execution of the program is immediately halted.

`-funittest`
: Turns on compilation of `unittest` code, and turns on the `version(unittest)` identifier. This implies `-fassert`.

`-fversion=`*`value`*
: Turns on compilation of conditional `version` code into the program identified by any of the following values:

  '`ident`' Turns on compilation of `version` code identified by *ident*.

`-fno-weak-templates`
: Turns off emission of declarations that can be defined in multiple objects as weak symbols. The default is to emit all public symbols as weak, unless the target lacks support for weak symbols. Disabling this option means that common symbols are instead put in COMDAT or become private.

1.3 Options for Directory Search

These options specify directories to search for files, libraries, and other parts of the compiler:

`-I`*`dir`*
: Specify a directory to use when searching for imported modules at compile time. Multiple `-I` options can be used, and the paths are searched in the same order.

`-J`*`dir`*
: Specify a directory to use when searching for files in string imports at compile time. This switch is required in order to use `import(file)` expressions. Multiple `-J` options can be used, and the paths are searched in the same order.

`-L`*`dir`*
: When linking, specify a library search directory, as with `gcc`.

`-B`*`dir`*
: This option specifies where to find the executables, libraries, source files, and data files of the compiler itself, as with `gcc`.

`-fmodule-file=`*`module`*`=`*`spec`*
: This option manipulates file paths of imported modules, such that if an imported module matches all or the leftmost part of *module*, the file path in *spec* is used as the location to search for D sources. This is used when the source file path and names are not the same as the package and module hierarchy. Consider the following examples:

```
gdc test.d -fmodule-file=A.B=foo.d -fmodule-file=C=bar
```

This will tell the compiler to search in all import paths for the source file *foo.d* when importing *A.B*, and the directory *bar/* when importing *C*, as annotated in the following D code:

```
module test;
```

```
import A.B;     // Matches A.B, searches for foo.d
import C.D.E;   // Matches C, searches for bar/D/E.d
import A.B.C;   // No match, searches for A/B/C.d
```

`-imultilib` *`dir`*
: Use *dir* as a subdirectory of the gcc directory containing target-specific D sources and interfaces.

`-iprefix` *`prefix`*
: Specify *prefix* as the prefix for the gcc directory containing target-specific D sources and interfaces. If the *prefix* represents a directory, you should include the final `'/'`.

`-nostdinc`
: Do not search the standard system directories for D source and interface files. Only the directories that have been specified with `-I` options (and the directory of the current file, if appropriate) are searched.

1.4 Code Generation

In addition to the many `gcc` options controlling code generation, `gdc` has several options specific to itself.

`-H`
: Generates D interface files for all modules being compiled. The compiler determines the output file based on the name of the input file, removes any directory components and suffix, and applies the `.di` suffix.

`-Hd` *`dir`*
: Same as `-H`, but writes interface files to directory *dir*. This option can be used with `-Hf` *`file`* to independently set the output file and directory path.

`-Hf` *`file`*
: Same as `-H` but writes interface files to *file*. This option can be used with `-Hd` *`dir`* to independently set the output file and directory path.

`-M`
: Output the module dependencies of all source files being compiled in a format suitable for `make`. The compiler outputs one `make` rule containing the object file name for that source file, a colon, and the names of all imported files.

`-MM`
: Like `-M` but does not mention imported modules from the D standard library package directories.

`-MF` *`file`*
: When used with `-M` or `-MM`, specifies a *file* to write the dependencies to. When used with the driver options `-MD` or `-MMD`, `-MF` overrides the default dependency output file.

`-MG`
: This option is for compatibility with `gcc`, and is ignored by the compiler.

`-MP`
: Outputs a phony target for each dependency other than the modules being compiled, causing each to depend on nothing.

`-MT` *`target`*
: Change the *target* of the rule emitted by dependency generation to be exactly the string you specify. If you want multiple targets, you can specify them as a single argument to `-MT`, or use multiple `-MT` options.

`-MQ target`
: Same as `-MT`, but it quotes any characters which are special to `make`.

`-MD`
: This option is equivalent to `-M -MF file`. The driver determines *file* by removing any directory components and suffix from the input file, and then adding a `.deps` suffix.

`-MMD`
: Like `-MD` but does not mention imported modules from the D standard library package directories.

`-X`
: Output information describing the contents of all source files being compiled in JSON format to a file. The driver determines *file* by removing any directory components and suffix from the input file, and then adding a `.json` suffix.

`-Xf file`
: Same as `-X`, but writes all JSON contents to the specified *file*.

`-fdoc`
: Generates `Ddoc` documentation and writes it to a file. The compiler determines *file* by removing any directory components and suffix from the input file, and then adding a `.html` suffix.

`-fdoc-dir=dir`
: Same as `-fdoc`, but writes documentation to directory *dir*. This option can be used with `-fdoc-file=file` to independently set the output file and directory path.

`-fdoc-file=file`
: Same as `-fdoc`, but writes documentation to *file*. This option can be used with `-fdoc-dir=dir` to independently set the output file and directory path.

`-fdoc-inc=file`
: Specify *file* as a *Ddoc* macro file to be read. Multiple `-fdoc-inc` options can be used, and files are read and processed in the same order.

`-fdump-c++-spec=file`
: For D source files, generate corresponding C++ declarations in *file*.

`-fdump-c++-spec-verbose`
: In conjunction with `-fdump-c++-spec=` above, add comments for ignored declarations in the generated C++ header.

`-fsave-mixins=file`
: Generates code expanded from D `mixin` statements and writes the processed sources to *file*. This is useful to debug errors in compilation and provides source for debuggers to show when requested.

1.5 Warnings

Warnings are diagnostic messages that report constructions that are not inherently erroneous but that 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.

`-Wall`
: Turns on all warnings messages. Warnings are not a defined part of the D language, and all constructs for which this may generate a warning message are valid code.

`-Walloca` This option warns on all uses of "alloca" in the source.

`-Walloca-larger-than=`*n*
: Warn on unbounded uses of alloca, and on bounded uses of alloca whose bound can be larger than *n* bytes. `-Wno-alloca-larger-than` disables `-Walloca-larger-than` warning and is equivalent to `-Walloca-larger-than=`*SIZE_MAX* or larger.

`-Wno-builtin-declaration-mismatch`
: Warn if a built-in function is declared with an incompatible signature.

`-Wcast-result`
: Warn about casts that will produce a null or zero result. Currently this is only done for casting between an imaginary and non-imaginary data type, or casting between a D and C++ class.

`-Wno-deprecated`
: Do not warn about usage of deprecated features and symbols with `deprecated` attributes.

`-Werror` Turns all warnings into errors.

`-Wextra` This enables some extra warning flags that are not enabled by `-Wall`.

```
-Waddress
-Wcast-result
-Wmismatched-special-enum
-Wunknown-pragmas
```

`-Wmismatched-special-enum`
: Warn when an enum the compiler recognizes as special is declared with a different size to the built-in type it is representing.

`-Wspeculative`
: List all error messages from speculative compiles, such as `__traits(compiles, ...)`. This option does not report messages as warnings, and these messages therefore never become errors when the `-Werror` option is also used.

`-Wunknown-pragmas`
: Warn when a `pragma()` is encountered that is not understood by `gdc`. This differs from `-fignore-unknown-pragmas` where a pragma that is part of the D language, but not implemented by the compiler, won't get reported.

`-Wno-varargs`
: Do not warn upon questionable usage of the macros used to handle variable arguments like `va_start`.

`-fno-ignore-unknown-pragmas`
: Do not recognize unsupported pragmas. Any `pragma()` encountered that is not part of the D language will result in an error. This option is now deprecated and will be removed in a future release.

`-fmax-errors=`*n*
: Limits the maximum number of error messages to *n*, at which point `gdc` bails out rather than attempting to continue processing the source code. If *n* is 0 (the default), 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 can be used in conjunction with `-fdoc` or `-H` to generate files for each module present on the command-line, but no other output file.

`-ftransition=`*`id`*
: Report additional information about D language changes identified by *id*. The following values are supported:

  '`all`' List information on all D language transitions.

  '`complex`' List all usages of complex or imaginary types.

  '`field`' List all non-mutable fields which occupy an object instance.

  '`in`' List all usages of `in` on parameter.

  '`nogc`' List all hidden GC allocations.

  '`templates`'
  List statistics on template instantiations.

  '`tls`' List all variables going into thread local storage.

1.6 Options for Linking

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.

`-defaultlib=`*`libname`*
: Specify the library to use instead of libphobos when linking. Options specifying the linkage of libphobos, such as `-static-libphobos` or `-shared-libphobos`, are ignored.

`-debuglib=`*`libname`*
: Specify the debug library to use instead of libphobos when linking. This option has no effect unless the `-g` option was also given on the command line. Options specifying the linkage of libphobos, such as `-static-libphobos` or `-shared-libphobos`, are ignored.

`-nophoboslib`
: Do not use the Phobos or D runtime library when linking. Options specifying the linkage of libphobos, such as `-static-libphobos` or `-shared-libphobos`, are ignored. The standard system libraries are used normally, unless `-nostdlib` or `-nodefaultlibs` is used.

`-shared-libphobos`
: On systems that provide `libgphobos` and `libgdruntime` as a shared and a static library, this option forces the use of the shared version. If no shared version was built when the compiler was configured, this option has no effect.

`-static-libphobos`
: On systems that provide `libgphobos` and `libgdruntime` as a shared and a static library, this option forces the use of the static version. If no static version was built when the compiler was configured, this option has no effect.

1.7 Developer Options

This section describes command-line options that are primarily of interest to developers or language tooling.

`-fdump-d-original`
: Output the internal front-end AST after the `semantic3` stage. This option is only useful for debugging the GNU D compiler itself.

`-v`
: Dump information about the compiler language processing stages as the source program is being compiled. This includes listing all modules that are processed through the `parse`, `semantic`, `semantic2`, and `semantic3` stages; all `import` modules and their file paths; and all `function` bodies that are being compiled.

2 Language Reference

The implementation of the D programming language used by the GNU D compiler is shared with parts of the front-end for the Digital Mars D compiler, hosted at `https://github.com/dlang/dmd/`. This common front-end covers lexical analysis, parsing, and semantic analysis of the D programming language defined in the documents at `https://dlang.org/`
.

The implementation details described in this manual are GNU D extensions to the D programming language. If you want to write code that checks whether these features are available, you can test for the predefined version `GNU`, or you can check whether a specific feature is compilable using `__traits(compiles)`.

```
version (GNU)
{
    import gcc.builtins;
    return __builtin_atan2(x, y);
}

static if (__traits(compiles, { asm {"";} }))
{
    asm { "magic instruction"; }
}
```

2.1 Attributes

User-Defined Attributes (UDA) are compile-time expressions introduced by the `@` token that can be attached to a declaration. These attributes can then be queried, extracted, and manipulated at compile time.

GNU D provides a number of extra special attributes to control specific compiler behavior that may help the compiler optimize or check code more carefully for correctness. The attributes are defined in the `gcc.attributes` module.

There is some overlap between the purposes of attributes and pragmas. It has been found more convenient to use `@attribute` to achieve a natural attachment of attributes to their corresponding declarations, whereas `pragma` is of use for compatibility with other compilers or constructs that do not naturally form part of the grammar.

2.1.1 Attribute Syntax

`@(gcc.attributes.attribute)` is the generic entrypoint for applying GCC attributes to a function, variable, or type. There is no type checking done, as well as no deprecation path for attributes removed from the compiler. So the recommendation is to use any of the other UDAs available as described in Section 2.1.2 [Common Attributes], page 12, unless it is a target-specific attribute (See Section 2.1.4 [Target Attributes], page 17).

Function attributes introduced by the `@attribute` UDA are used in the declaration of a function, followed by an attribute name string and any arguments separated by commas enclosed in parentheses.

```
import gcc.attributes;
@attribute("regparm", 1) int func(int size);
```

Multiple attributes can be applied to a single declaration either with multiple `@attribute` attributes, or passing all attributes as a comma-separated list enclosed by parentheses.

```
// Both func1 and func2 have the same attributes applied.
```

```
@attribute("noinline") @attribute("noclone") void func1();
@(attribute("noinline"), attribute("noclone")) void func2();
```

There are some problems with the semantics of such attributes in D. For example, there are no manglings for attributes, although they may affect code generation, so problems may arise when attributed types are used in conjunction with templates or overloading. Similarly, `typeid` does not distinguish between types with different attributes. Support for attributes in D are restricted to declarations only.

2.1.2 Common Attributes

The following attributes are supported on most targets.

`@(gcc.attributes.alloc_size (`*`sizeArgIdx`*`))`
`@(gcc.attributes.alloc_size (`*`sizeArgIdx, numArgIdx`*`))`
`@(gcc.attributes.alloc_size (`*`sizeArgIdx, numArgIdx, zeroBasedNumbering`*`))`

The `@alloc_size` attribute may be applied to a function - or a function pointer variable - that returns a pointer and takes at least one argument of an integer or enumerated type. It indicates that the returned pointer points to memory whose size is given by the function argument at `sizeArgIdx`, or by the product of the arguments at `sizeArgIdx` and `numArgIdx`. Meaningful sizes are positive values less than `ptrdiff_t.max`. Unless `zeroBasedNumbering` is true, argument numbering starts at one for ordinary functions, and at two for non-static member functions.

If `numArgIdx` is less than `0`, it is taken to mean there is no argument specifying the element count.

```
@alloc_size(1) void* malloc(size_t);
@alloc_size(3,2) void* reallocarray(void *, size_t, size_t);
@alloc_size(1,2) void* my_calloc(size_t, size_t, bool);
void malloc_cb(@alloc_size(1) void* function(size_t) ptr) { }
```

`@(gcc.attributes.always_inline)`

The `@always_inline` attribute inlines the function independent of any restrictions that otherwise apply to inlining. Failure to inline such a function is diagnosed as an error.

```
@always_inline int func();
```

`@(gcc.attributes.cold)`

The `@cold` attribute on functions is used to inform the compiler that the function is unlikely to be executed. The function is optimized for size rather than speed and on many targets it is placed into a special subsection of the text section so all cold functions appear close together, improving code locality of non-cold parts of program. The paths leading to calls of cold functions within code are considered to be cold too.

```
@cold int func();
```

`@(gcc.attributes.flatten)`

The `@flatten` attribute is used to inform the compiler that every call inside this function should be inlined, if possible. Functions declared with attribute `@noinline` and similar are not inlined.

```
@flatten int func();
```

`@(gcc.attributes.no_icf)`

The `@no_icf` attribute prevents a function from being merged with another semantically equivalent function.

```
@no_icf int func();
```

`@(gcc.attributes.no_sanitize ("`*`sanitize_option`*`"))`

The `@no_sanitize` attribute on functions is used to inform the compiler that it should not do sanitization of any option mentioned in *sanitize_option*. A list of values acceptable by the `-fsanitize` option can be provided.

```
@no_sanitize("alignment", "object-size") void func1() { }
@no_sanitize("alignment,object-size") void func2() { }
```

`@(gcc.attributes.noclone)`

The `@noclone` attribute prevents a function from being considered for cloning - a mechanism that produces specialized copies of functions and which is (currently) performed by interprocedural constant propagation.

```
@noclone int func();
```

`@(gcc.attributes.noinline)`

The `@noinline` attribute prevents a function from being considered for inlining. If the function does not have side effects, there are optimizations other than inlining that cause function calls to be optimized away, although the function call is live. To keep such calls from being optimized away, put `asm { ""; }` in the called function, to serve as a special side effect.

```
@noinline int func();
```

`@(gcc.attributes.noipa)`

The `@noipa` attribute disables interprocedural optimizations between the function with this attribute and its callers, as if the body of the function is not available when optimizing callers and the callers are unavailable when optimizing the body. This attribute implies `@noinline`, `@noclone`, and `@no_icf` attributes. However, this attribute is not equivalent to a combination of other attributes, because its purpose is to suppress existing and future optimizations employing interprocedural analysis, including those that do not have an attribute suitable for disabling them individually.

This attribute is supported mainly for the purpose of testing the compiler.

```
@noipa int func();
```

`@(gcc.attributes.noplt)`

The `@noplt` attribute is the counterpart to option `-fno-plt`. Calls to functions marked with this attribute in position-independent code do not use the PLT in position-independent code.

In position-dependant code, a few targets also convert call to functions that are marked to not use the PLT to use the GOT instead.

```
@noplt int func();
```

`@(gcc.attributes.optimize (`*`arguments`*`))`

The `@optimize` attribute is used to specify that a function is to be compiled with different optimization options than specified on the command line. Valid *arguments* are constant non-negative integers and strings. Multiple arguments can be provided, separated by commas to specify multiple options. Each numeric argument specifies an optimization level. Each string argument that begins with the letter `O` refers to an optimization option such as `-O0` or `-Os`. Other options are taken as suffixes to the `-f` prefix jointly forming the name of an optimization option.

Not every optimization option that starts with the `-f` prefix specified by the attribute necessarily has an effect on the function. The `@optimize` attribute should be used for debugging purposes only. It is not suitable in production code.

```
@optimize(2) double fn0(double x);
@optimize("2") double fn1(double x);
@optimize("s") double fn2(double x);
@optimize("Ofast") double fn3(double x);
@optimize("-O2") double fn4(double x);
@optimize("tree-vectorize") double fn5(double x);
@optimize("-ftree-vectorize") double fn6(double x);
@optimize("no-finite-math-only", 3) double fn7(double x);
```

`@(gcc.attributes.register ("`*registerName*`"))`

The `@register` attribute specifies that a local or `__gshared` variable is to be given a register storage-class in the C99 sense of the term, and will be placed into a register named *registerName*.

The variable needs to boiled down to a data type that fits the target register. It also cannot have either thread-local or `extern` storage. It is an error to take the address of a register variable.

```
@register("ebx") __gshared int ebx = void;
void func() { @register("r10") long r10 = 0x2a; }
```

`@(gcc.attributes.restrict)`

The `@restrict` attribute specifies that a function parameter is to be restrict-qualified in the C99 sense of the term. The parameter needs to boil down to either a pointer or reference type, such as a D pointer, class reference, or a `ref` parameter.

```
void func(@restrict ref const float[16] array);
```

`@(gcc.attributes.section ("`*sectionName*`"))`

The `@section` attribute specifies that a function or variable lives in a particular section. For when you need certain particular functions to appear in special sections.

Some file formats do not support arbitrary sections so the section attribute is not available on all platforms. If you need to map the entire contents of a module to a particular section, consider using the facilities of the linker instead.

```
@section("bar") extern void func();
@section("stack") ubyte[10000] stack;
```

`@(gcc.attributes.simd)`

The `@simd` attribute enables creation of one or more function versions that can process multiple arguments using SIMD instructions from a single invocation. Specifying this attribute allows compiler to assume that such versions are available at link time (provided in the same or another module). Generated versions are target-dependent and described in the corresponding Vector ABI document.

```
@simd double sqrt(double x);
```

`@(gcc.attributes.simd_clones ("`*`mask`*`"))`

The `@simd_clones` attribute is the same as `@simd`, but also includes a *mask* argument. Valid masks values are `notinbranch` or `inbranch`, and instructs the compiler to generate non-masked or masked clones correspondingly.

```
@simd_clones("notinbranch") double atan2(double y, double x);
```

`@(gcc.attributes.symver ("`*`arguments`*`"))`

The `@symver` attribute creates a symbol version on ELF targets. The syntax of the string parameter is `"`*`name@nodename`*`"`. The *name* part of the parameter is the actual name of the symbol by which it will be externally referenced. The *nodename* portion should be the name of a node specified in the version script supplied to the linker when building a shared library. Versioned symbol must be defined and must be exported with default visibility.

Finally if the parameter is `"`*`name@@nodename`*`"` then in addition to creating a symbol version (as if `"`*`name@nodename`*`"` was used) the version will be also used to resolve *name* by the linker.

```
@symver("foo@VERS_1") int foo_v1();
```

`@(gcc.attributes.target ("`*`options`*`"))`

The `@target` attribute is used to specify that a function is to be compiled with different target options than specified on the command line. One or more strings can be provided as arguments, separated by commas to specify multiple options. Each string consists of one or more comma-separated suffixes to the `-m` prefix jointly forming the name of a machine-dependent option.

The target attribute can be used for instance to have a function compiled with a different ISA (instruction set architecture) than the default.

The options supported are specific to each target.

```
@target("arch=core2") void core2_func();
@target("sse3") void sse3_func();
```

`@(gcc.attributes.target_clones ("`*`options`*`"))`

The `@target_clones` attribute is used to specify that a function be cloned into multiple versions compiled with different target *options* than specified on the command line. The supported options and restrictions are the same as for `@target` attribute.

It also creates a resolver function that dynamically selects a clone suitable for current architecture. The resolver is created only if there is a usage of a function with `@target_clones` attribute.

```
@target_clones("sse4.1,avx,default") double func(double x);
```

`@(gcc.attributes.used)`

The `@used` attribute, annotated to a function or variable, means that code must be emitted for the function even if it appears that the function is not referenced. This is useful, for example, when the function is referenced only in inline assembly.

```
@used __gshared int var = 0x1000;
```

`@(gcc.attributes.visibility ("`*`visibilityName`*`"))`

The `@visibility` attribute affects the linkage of the declaration to which it is attached. It can be applied to variables, types, and functions.

There are four supported visibility_type values: `default`, `hidden`, `protected`, or `internal` visibility.

```
@visibility("protected") void func() {  }
```

`@(gcc.attributes.weak)`

The `@weak` attribute causes a declaration of an external symbol to be emitted as a weak symbol rather than a global. This is primarily useful in defining library functions that can be overridden in user code, though it can also be used with non-function declarations. The overriding symbol must have the same type as the weak symbol. In addition, if it designates a variable it must also have the same size and alignment as the weak symbol.

Weak symbols are supported for ELF targets, and also for a.out targets when using the GNU assembler and linker.

```
@weak int func() { return 1; }
```

2.1.3 Other Attributes

The following attributes are defined for compatibility with other compilers.

`@(gcc.attributes.allocSize (`*`sizeArgIdx`*`))`
`@(gcc.attributes.allocSize (`*`sizeArgIdx`*`, `*`numArgIdx`*`))`
`@(gcc.attributes.allocSize (`*`sizeArgIdx`*`))`

These attributes are a synonym for `@alloc_size(`*`sizeArgIdx`*`, `*`numArgIdx`*`, true)`. Unlike `@alloc_size`, it uses 0-based index of the function arguments.

`@(gcc.attributes.assumeUsed)`

This attribute is a synonym for `@used`.

`@(gcc.attributes.dynamicCompile)`
`@(gcc.attributes.dynamicCompileConst)`
`@(gcc.attributes.dynamicCompileEmit)`

These attributes are accepted, but have no effect.

`@(gcc.attributes.fastmath)`

This attribute is a synonym for `@optimize("Ofast")`. Explicitly sets "fast-math" for a function, enabling aggressive math optimizations.

`@(gcc.attributes.hidden)`

This attribute is a synonym for `@visibility("hidden")`. Sets the visibility of a function or global variable to "hidden".

`@(gcc.attributes.naked)`

This attribute is a synonym for `@attribute("naked")`. Adds GCC's "naked" attribute to a function, disabling function prologue / epilogue emission. Intended to be used in combination with basic `asm` statements. While using extended `asm` or a mixture of basic `asm` and D code may appear to work, they cannot be depended upon to work reliably and are not supported.

`@(gcc.attributes.noSanitize ("`*`sanitize_option`*`"))`
: This attribute is a synonym for `@no_sanitize("sanitize_option")`.

`@(gcc.attributes.optStrategy ("`*`strategy`*`"))`
: This attribute is a synonym for `@optimize("O0")` and `@optimize("Os")`. Sets the optimization strategy for a function. Valid strategies are "none", "optsize", "minsize". The strategies are mutually exclusive.

`@(gcc.attributes.polly)`
: This attribute is a synonym for `@optimize("loop-parallelize-all", "loop-nest-optimize")`. Only effective when GDC was built with ISL included.

2.1.4 Target-specific Attributes

Many targets have their own target-specific attributes. These are also exposed via the `gcc.attributes` module with use of the generic `@(gcc.attributes.attribute)` UDA function.

See Section 2.1.1 [Attribute Syntax], page 11, for details of the exact syntax for using attributes.

See the function and variable attribute documentation in the GCC manual for more information about what attributes are available on each target.

Examples of using x86-specific target attributes are shown as follows:

```
import gcc.attributes;

@attribute("cdecl")
@attribute("fastcall")
@attribute("ms_abi")
@attribute("sysv_abi")
@attribute("callee_pop_aggregate_return", 1)
@attribute("ms_hook_prologue")
@attribute("naked")
@attribute("regparm", 2)
@attribute("sseregparm")
@attribute("force_align_arg_pointer")
@attribute("stdcall")
@attribute("no_caller_saved_registers")
@attribute("interrupt")
@attribute("indirect_branch", "thunk")
@attribute("function_return", "keep"))
@attribute("nocf_check")
@attribute("cf_check")
@attribute("indirect_return")
@attribute("fentry_name", "nop")
@attribute("fentry_section", "__entry_loc")
@attribute("nodirect_extern_access")
```

2.2 Built-in Functions

GCC provides a large number of built-in functions that are made available in GNU D by importing the `gcc.builtins` module. Declarations in this module are automatically created by the compiler. All declarations start with `__builtin_`. Refer to the built-in function documentation in the GCC manual for a full list of functions that are available.

2.2.1 Built-in Types

In addition to built-in functions, the following types are defined in the `gcc.builtins` module.

`___builtin_clong`
: The D equivalent of the target's C `long` type.

`___builtin_clonglong`
: The D equivalent of the target's C `long long` type.

`___builtin_culong`
: The D equivalent of the target's C `unsigned long` type.

`___builtin_culonglong`
: The D equivalent of the target's C `unsigned long long` type.

`___builtin_machine_byte`
: Signed unit-sized integer type.

`___builtin_machine_int`
: Signed word-sized integer type.

`___builtin_machine_ubyte`
: Unsigned unit-sized integer type.

`___builtin_machine_uint`
: Unsigned word-sized integer type.

`___builtin_pointer_int`
: Signed pointer-sized integer type.

`___builtin_pointer_uint`
: Unsigned pointer-sized integer type.

`___builtin_unwind_int`
: The D equivalent of the target's C `_Unwind_Sword` type.

`___builtin_unwind_uint`
: The D equivalent of the target's C `_Unwind_Word` type.

`___builtin_va_list`
: The target's `va_list` type.

2.2.2 Querying Available Built-ins

Not all of the functions are supported, and some target-specific functions may only be available when compiling for a particular ISA. One way of finding out what is exposed by the built-ins module is by generating a D interface file. Assuming you have no file `builtins.d`, the command

```
echo "module gcc.builtins;" > builtins.d; gdc -H -fsyntax-only builtins.d
```

will save all built-in declarations to the file `builtins.di`.

Another way to determine whether a specific built-in is available is by using compile-time reflection.

```
enum X86_HAVE_SSE3 = __traits(compiles, __builtin_ia32_haddps);
```

```
enum X86_HAVE_SSSE3 = __traits(compiles, __builtin_ia32_pmulhrsw128);
enum X86_HAVE_SSE41 = __traits(compiles, __builtin_ia32_dpps);
enum X86_HAVE_SSE42 = __traits(compiles, __builtin_ia32_pcmpgtq);
enum X86_HAVE_AVX = __traits(compiles, __builtin_ia32_vbroadcastf128_pd256);
enum X86_HAVE_AVX2 = __traits(compiles, __builtin_ia32_gathersiv2df);
enum X86_HAVE_BMI2 = __traits(compiles, __builtin_ia32_pext_si);
```

2.2.3 Other Built-in Functions

As well as built-ins being available from the `gcc.builtins` module, GNU D will also recognize when an `extern(C)` library function is a GCC built-in. Many of these functions are only optimized in certain cases; if they are not optimized in a particular case, a call to the library function is emitted. This optimization can be disabled with the `-fno-builtin` option (see Section 1.2 [Runtime Options], page 1).

In the `core.stdc.complex` module, the functions `cabs`, `cabsf`, `cabsl`, `cacos`, `cacosf`, `cacosh`, `cacoshf`, `cacoshl`, `cacosl`, `carg`, `cargf`, `cargl`, `casin`, `casinf`, `casinh`, `casinhf`, `casinhl`, `casinl`, `catan`, `catanf`, `catanh`, `catanhf`, `catanhl`, `catanl`, `ccos`, `ccosf`, `ccosh`, `ccoshf`, `ccoshl`, `ccosl`, `cexp`, `cexpf`, `cexpl`, `clog`, `clogf`, `clogl`, `conj`, `conjf`, `conjl`, `cpow`, `cpowf`, `cpowl`, `cproj`, `cprojf`, `cprojl`, `csin`, `csinf`, `csinh`, `csinhf`, `csinhl`, `csinl`, `csqrt`, `csqrtf`, `csqrtl`, `ctan`, `ctanf`, `ctanh`, `ctanhf`, `ctanhl`, `ctanl` may be handled as built-in functions. All these functions have corresponding versions prefixed with `__builtin_` in the `gcc.builtins` module.

In the `core.stdc.ctype` module, the functions `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`, `tolower`, `toupper` may be handled as built-in functions. All these functions have corresponding versions prefixed with `__builtin_` in the `gcc.builtins` module.

In the `core.stdc.fenv` module, the functions `feclearexcept`, `fegetenv`, `fegetexceptflag`, `fegetround`, `feholdexcept`, `feraiseexcept`, `fesetenv`, `fesetexceptflag`, `fesetround`, `fetestexcept`, `feupdateenv` may be handled as built-in functions. All these functions have corresponding versions prefixed with `__builtin_` in the `gcc.builtins` module.

In the `core.stdc.inttypes` module, the function `imaxabs` may be handled as a built-in function. All these functions have corresponding versions prefixed with `__builtin_` in the `gcc.builtins` module.

In the `core.stdc.math` module, the functions `acos`, `acosf`, `acosh`, `acoshf`, `acoshl`, `acosl`, `asin`, `asinf`, `asinh`, `asinhf`, `asinhl`, `asinl`, `atan`, `atan2`, `atan2f`, `atan2l`, `atanf`, `atanh`, `atanhf`, `atanhl`, `atanl`, `cbrt`, `cbrtf`, `cbrtl`, `ceil`, `ceilf`, `ceill`, `copysign`, `copysignf`, `copysignl`, `cos`, `cosf`, `cosh`, `coshf`, `coshl`, `cosl`, `erf`, `erfc`, `erfcf`, `erfcl`, `erff`, `erfl`, `exp`, `exp2`, `exp2f`, `exp2l`, `expf`, `expl`, `expm1`, `expm1f`, `expm1l`, `fabs`, `fabsf`, `fabsl`, `fdim`, `fdimf`, `fdiml`, `floor`, `floorf`, `floorl`, `fma`, `fmaf`, `fmal`, `fmax`, `fmaxf`, `fmaxl`, `fmin`, `fminf`, `fminl`, `fmod`, `fmodf`, `fmodl`, `frexp`, `frexpf`, `frexpl`, `hypot`, `hypotf`, `hypotl`, `ilogb`, `ilogbf`, `ilogbl`, `isinf`, `isnan`, `ldexp`, `ldexpf`, `ldexpl`, `lgamma`, `lgammaf`, `lgammal`, `llrint`, `llrintf`, `llrintl`, `llround`, `llroundf`, `llroundl`, `log`, `log10`, `log10f`, `log10l`, `log1p`, `log1pf`, `log1pl`, `log2`, `log2f`, `log2l`, `logb`, `logbf`, `logbl`, `logf`, `logl`, `lrint`, `lrintf`, `lrintl`, `lround`, `lroundf`, `lroundl`, `modf`, `modff`, `modfl`, `nan`, `nanf`, `nanl`, `nearbyint`, `nearbyintf`, `nearbyintl`, `nextafter`, `nextafterf`, `nextafterl`, `nexttoward`, `nexttowardf`, `nexttowardl`, `pow`, `powf`, `powl`, `remainder`, `remainderf`, `remainderl`, `remquo`, `remquof`, `remquol`, `rint`, `rintf`, `rintl`, `round`, `roundf`, `roundl`, `scalbln`, scalblnf, scalblnl, scalbn, scalbnf, scalbnl, signbit, sin, sinf, sinh, sinhf, sinhl, sinl, sqrt, sqrtf, sqrtl, tan, tanf, tanh, tanhf, tanhl, tanl, tgamma, tgammaf, tgammal, trunc, truncf, truncl may be handled as built-in functions. All these functions have corresponding versions prefixed with __builtin_ in the gcc.builtins module.

In the core.stdc.stdio module, the functions fprintf, fputc, fputc_unlocked, fputs, fwrite, printf, puts, snprintf, sprintf, vfprintf, vprintf, vsnprintf, vsprintf may be handled as built-in functions. All these functions have corresponding versions prefixed with __builtin_ in the gcc.builtins module.

In the core.stdc.stdlib module, the functions abort, abs, aligned_alloc, alloca, calloc, exit, _Exit, free, labs, llabs, malloc, realloc may be handled as built-in functions. All these functions have corresponding versions prefixed with __builtin_ in the gcc.builtins module.

In the core.stdc.string module, the functions memchr, memcmp, memcpy, memmove, memset, strcat, strchr, strcmp, strcpy, strcspn, strdup, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr may be handled as built-in functions. All these functions have corresponding versions prefixed with __builtin_ in the gcc.builtins module.

In the core.stdc.time module, the function strftime may be handled as a built-in function. All these functions have corresponding versions prefixed with __builtin_ in the gcc.builtins module.

In the core.stdc.wctype module, the functions iswalnum, iswalpha, iswblank, iswcntrl, iswdigit, iswgraph, iswlower, iswprint, iswpunct, iswspace, iswupper, iswxdigit, towlower, towupper may be handled as built-in functions. All these functions have corresponding versions prefixed with __builtin_ in the gcc.builtins module.

Within the core.sys package for POSIX and platform definitions, the functions putchar_unlocked, putc_unlocked, posix_memalign, ffs, strcasecmp, strncasecmp, stpcpy, stpncpy, strndup, strnlen, execl, execle, execlp, execv, execve, execvp, _exit, fork may be handled as built-in functions. All these functions have corresponding versions prefixed with __builtin_ in the gcc.builtins module.

2.3 Importing C Sources into D

ImportC is a C preprocessor and parser embedded into the GNU D implementation. It enables direct importation of C files, without needing to manually prepare a D file corresponding to the declarations in the C file.

ImportC is an implementation of ISO/IEC 9899:2011, which will be referred to as C11. Prior versions, such as C99, C89, and K+R C, are not supported.

Assuming you have no file cstdio.c or main.d, the commands

```
cat > cstdio.c << @EOC
int printf(const char*, ...);
@EOC
cat > main.d << @EOD
import cstdio;
void main() { printf("Hello ImportC\n"); }
@EOD
gdc main.d -o main; ./main
```

will generate a program which will print 'Hello ImportC'.

ImportC does not have a preprocessor. It is designed to compile C files after they have been first run through the C preprocessor. If the C file has a '.i' extension, the file is presumed to be already preprocessed. Preprocessing can be run manually:

```
gcc -E file.c > file.i
```

ImportC collects all the `#define` macros from the preprocessor run when it is run automatically. The macros that look like manifest constants, such as:

```
#define COLOR 0x123456
```

are interpreted as D manifest constant declarations of the form:

```
enum COLOR = 0x123456;
```

The variety of macros that can be interpreted as D declarations may be expanded, but will never encompass all the metaprogramming uses of C macros.

GNU D does not directly compile C files into modules that can be linked in with D code to form an executable. When given a source file with the suffix '.c', the compiler driver program `gdc` instead runs the subprogram `cc1`.

```
gdc file1.d file2.c // d21 file1.d -o file1.s
                    // cc1 file2.c -o file2.s
                    // as file1.s -o file1.o
                    // as file2.s -o file2.o
                    // ld file1.o file2.o
```

2.4 Inline Assembly

The `asm` keyword allows you to embed assembler instructions within D code. GNU D provides two forms of inline `asm` statements. A *basic* `asm` statement is one with no operands, while an *extended* `asm` statement includes one or more operands.

```
asm FunctionAttributes {
        AssemblerInstruction ;
}

asm FunctionAttributes {
        AssemblerTemplate
         : OutputOperands
         [ : InputOperands
         [ : Clobbers
         [ : GotoLabels ] ] ] ;
}
```

The extended form is preferred for mixing D and assembly language within a function, but to include assembly language in a function declared with the `naked` attribute you must use basic `asm`.

```
uint incr (uint value)
{
    uint result;
    asm { "incl %0"
          : "=a" (result)
          : "a" (value);
    }
    return result;
}
```

Multiple assembler instructions can appear within an `asm` block, or the instruction template can be a multi-line or concatenated string. In both cases, GCC's optimizers won't discard or move any instruction within the statement block.

```
bool hasCPUID()
{
    uint flags = void;
    asm nothrow @nogc {
        "pushfl";
        "pushfl";
        "xorl %0, (%%esp)" :: "i" (0x00200000);
        "popfl";
        "pushfl";
        "popl %0" : "=a" (flags);
        "xorl (%%esp), %0" : "=a" (flags);
        "popfl";
    }
    return (flags & 0x0020_0000) != 0;
}
```

The instruction templates for both basic and extended `asm` can be any expression that can be evaluated at compile-time to a string, not just string literals.

```
uint invert(uint v)
{
    uint result;
    asm @safe @nogc nothrow pure {
        genAsmInsn(`invert`)
        : [res] `=r` (result)
        : [arg1] `r` (v);
    }
    return result;

}
```

The total number of input + output + goto operands is limited to 30.

2.5 Intrinsics

The D language specification itself does not define any intrinsics that a compatible compiler must implement. Rather, within the D core library there are a number of modules that define primitives with generic implementations. While the generic versions of these functions are computationally expensive relative to the cost of the operation itself, compiler implementations are free to recognize them and generate equivalent and faster code.

The following are the kinds of intrinsics recognized by GNU D.

2.5.1 Bit Operation Intrinsics

The following functions are a collection of intrinsics that do bit-level operations, available by importing the `core.bitop` module.

Although most are named after x86 hardware instructions, it is not guaranteed that they will result in generating equivalent assembly on x86. If the compiler determines there is a better way to get the same result in hardware, then that will be used instead.

`int core.bitop.bsf (uint v)` [Function]
`int core.bitop.bsf (ulong v)` [Function]

Scans the bits in *v* starting with bit `0`, looking for the first set bit. Returns the bit number of the first bit set. The return value is undefined if *v* is zero.

This intrinsic is the same as the GCC built-in function `__builtin_ctz`.

`int core.bitop.bsr (uint v)` [Function]
`int core.bitop.bsr (ulong v)` [Function]

Scans the bits in *v* from the most significant bit to the least significant bit, looking for the first set bit. Returns the bit number of the first bit set. The return value is undefined if *v* is zero.

This intrinsic is equivalent to writing the following:

```
result = __builtin_clz(v) ^ (v.sizeof * 8 - 1)
```

`int core.bitop.bt (scope const(uint*) p, uint bitnum)` [Function]
`int core.bitop.bt (scope const(uint*) p, uint bitnum)` [Function]

Tests the bit *bitnum* in the input parameter *p*. Returns a non-zero value if the bit was set, and a zero if it was clear.

This intrinsic is equivalent to writing the following:

```
immutable bits_per_unit = (*p).sizeof * 8;
immutable bit_mask = size_t(1) << (bitnum % bits_per_unit);

result = (p[bitnum / bits_per_unit] & bit_mask) != 0;
```

`int core.bitop.btc (uint* p, uint bitnum)` [Function]
`int core.bitop.btc (ulong* p, ulong bitnum)` [Function]

Tests and complements the bit *bitnum* in the input parameter *p*. Returns a non-zero value if the bit was set, and a zero if it was clear.

This intrinsic is equivalent to writing the following:

```
immutable bits_per_unit = (*p).sizeof * 8;
immutable bit_mask = size_t(1) << (bitnum % bits_per_unit);

result = (p[bitnum / bits_per_unit] & bit_mask) != 0;

p[bitnum / bits_per_unit] ^= bit_mask;
```

`int core.bitop.btr (uint* p, uint bitnum)` [Function]
`int core.bitop.btr (ulong* p, ulong bitnum)` [Function]

Tests and resets (sets to 0) the bit *bitnum* in the input parameter *p*. Returns a non-zero value if the bit was set, and a zero if it was clear.

This intrinsic is equivalent to writing the following:

```
immutable bits_per_unit = (*p).sizeof * 8;
immutable bit_mask = size_t(1) << (bitnum % bits_per_unit);

result = (p[bitnum / bits_per_unit] & bit_mask) != 0;

p[bitnum / bits_per_unit] &= ~bit_mask;
```

`int core.bitop.bts (uint* p, uint bitnum)` [Function]
`int core.bitop.bts (ulong* p, ulong bitnum)` [Function]

Tests and sets the bit *bitnum* in the input parameter *p*. Returns a non-zero value if the bit was set, and a zero if it was clear.

This intrinsic is equivalent to writing the following:

```
immutable bits_per_unit = (*p).sizeof * 8;
immutable bit_mask = size_t(1) << (bitnum % bits_per_unit);

result = (p[bitnum / bits_per_unit] & bit_mask) != 0;

p[bitnum / bits_per_unit] |= bit_mask;
```

`ushort core.bitop.byteswap (ushort x)` [Function]
`uint core.bitop.bswap (uint x)` [Function]
`ulong core.bitop.bswap (ulong x)` [Function]

Swaps the bytes in *x* end-to-end; for example, in a 4-byte `uint`, byte `0` becomes byte `3`, byte `1` becomes byte `2`, etc.

This intrinsic is the same as the GCC built-in function `__builtin_bswap`.

`int core.bitop.popcnt (uint x)` [Function]
`int core.bitop.popcnt (ulong x)` [Function]

Calculates the number of set bits in *x*.

This intrinsic is the same as the GCC built-in function `__builtin_popcount`.

`T core.bitop.rol (T)(const T value, const uint count)` [Template]
`T core.bitop.rol (uint count, T)(const T value)` [Template]

Bitwise rotate *value* left by *count* bit positions.

This intrinsic is equivalent to writing the following:

```
result = cast(T) ((value << count) | (value >> (T.sizeof * 8 - count)));
```

`T core.bitop.ror (T)(const T value, const uint count)` [Template]
`T core.bitop.ror (uint count, T)(const T value)` [Template]

Bitwise rotate *value* right by *count* bit positions.

This intrinsic is equivalent to writing the following:

```
result = cast(T) ((value >> count) | (value << (T.sizeof * 8 - count)));
```

2.5.2 Integer Overflow Intrinsics

The following functions are a collection of intrinsics that implement integral arithmetic primitives that check for out-of-range results, available by importing the `core.checkedint` module.

In all intrinsics, the overflow is sticky, meaning a sequence of operations can be done and overflow need only be checked at the end.

`int core.checkedint.adds (int x, int y, ref bool overflow)` [Function]
`long core.checkedint.adds (long x, long y, ref bool overflow)` [Function]

Add two signed integers, checking for overflow.

This intrinsic is the same as the GCC built-in function `__builtin_sadd_overflow`.

`int core.checkedint.addu (int x, int y, ref bool overflow)` [Function]
`long core.checkedint.addu (long x, long y, ref bool overflow)` [Function]

Add two unsigned integers, checking for overflow.

This intrinsic is the same as the GCC built-in function `__builtin_uadd_overflow`.

`int core.checkedint.muls (int x, int y, ref bool overflow)` [Function]
`long core.checkedint.muls (long x, long y, ref bool overflow)` [Function]

Multiply two signed integers, checking for overflow.

This intrinsic is the same as the GCC built-in function `__builtin_smul_overflow`.

`int core.checkedint.mulu (int x, int y, ref bool overflow)` [Function]
`long core.checkedint.mulu (long x, long y, ref bool overflow)` [Function]

Multiply two unsigned integers, checking for overflow.

This intrinsic is the same as the GCC built-in function `__builtin_umul_overflow`.

`int core.checkedint.negs (int x, ref bool overflow)` [Function]
`long core.checkedint.negs (long x, ref bool overflow)` [Function]

Negates an integer.

This intrinsic is equivalent to writing the following:

```
result = __builtin_ssub (0, x, overflow);
```

`int core.checkedint.subs (int x, int y, ref bool overflow)` [Function]
`long core.checkedint.subs (long x, long y, ref bool overflow)` [Function]

Subtract two signed integers, checking for overflow.

This intrinsic is the same as the GCC built-in function `__builtin_ssub_overflow`.

`int core.checkedint.subu (int x, int y, ref bool overflow)` [Function]
`long core.checkedint.subu (long x, long y, ref bool overflow)` [Function]

Subtract two unsigned integers, checking for overflow.

This intrinsic is the same as the GCC built-in function `__builtin_usub_overflow`.

2.5.3 Math Intrinsics

The following functions are a collection of mathematical intrinsics, available by importing the `core.math` module.

`float core.math.cos (float x)` [Function]
`double core.math.cos (double x)` [Function]
`real core.math.cos (real x)` [Function]

Returns cosine of *x*, where *x* is in radians. The return value is undefined if *x* is greater than 2^{64}.

This intrinsic is the same as the GCC built-in function `__builtin_cos`.

`float core.math.fabs (float x)` [Function]
`double core.math.fabs (double x)` [Function]
`real core.math.fabs (real x)` [Function]

Compute the absolute value of *x*.

This intrinsic is the same as the GCC built-in function `__builtin_fabs`.

`float core.math.ldexp (float n, int exp)` [Function]
`double core.math.ldexp (double n, int exp)` [Function]
`real core.math.ldexp (real n, int exp)` [Function]

Compute $n * 2^{exp}$.

This intrinsic is the same as the GCC built-in function `__builtin_ldexp`.

`float core.math.rint (float x)` [Function]
`double core.math.rint (double x)` [Function]
`real core.math.rint (real x)` [Function]

Rounds *x* to the nearest integer value, using the current rounding mode. If the return value is not equal to *x*, the `FE_INEXACT` exception is raised. `nearbyint` performs the same operation, but does not set the `FE_INEXACT` exception.

This intrinsic is the same as the GCC built-in function `__builtin_rint`.

`float core.math.rndtol (float x)` [Function]
`double core.math.rndtol (double x)` [Function]
`real core.math.rndtol (real x)` [Function]

Returns *x* rounded to a long value using the current rounding mode. If the integer value of *x* is greater than `long.max`, the result is indeterminate.

This intrinsic is the same as the GCC built-in function `__builtin_llround`.

`float core.math.sin (float x)` [Function]
`double core.math.sin (double x)` [Function]
`real core.math.sin (real x)` [Function]

Returns sine of *x*, where *x* is in radians. The return value is undefined if *x* is greater than 2^{64}.

This intrinsic is the same as the GCC built-in function `__builtin_sin`.

`float core.math.sqrt (float x)` [Function]
`double core.math.sqrt (double x)` [Function]
`real core.math.sqrt (real x)` [Function]

Compute the sqrt of *x*.

This intrinsic is the same as the GCC built-in function `__builtin_sqrt`.

`T core.math.toPrec (T)(float f)` [Template]
`T core.math.toPrec (T)(double f)` [Template]
`T core.math.toPrec (T)(real f)` [Template]

Round *f* to a specific precision.

In floating-point operations, D language types specify only a minimum precision, not a maximum. The `toPrec` function forces rounding of the argument *f* to the precision of the specified floating point type `T`. The rounding mode used is inevitably target-dependent, but will be done in a way to maximize accuracy. In most cases, the default is round-to-nearest.

2.5.4 Variadic Intrinsics

The following functions are a collection of variadic intrinsics, available by importing the `core.stdc.stdarg` module.

`void core.stdc.stdarg.va_arg (T)(ref va_list ap, ref T parmn)` [Template]

Retrieve and store in *parmn* the next value from the `va_list` *ap* that is of type `T`.

This intrinsic is equivalent to writing the following:

```
parmn = __builtin_va_arg (ap, T);
```

`T core.stdc.stdarg.va_arg (T)(ref va_list ap)` [Template]

Retrieve and return the next value from the `va_list` *ap* that is of type `T`.

This intrinsic is equivalent to writing the following:

```
result = __builtin_va_arg (ap, T);
```

`void core.stdc.stdarg.va_copy (out va_list dest, va_list src)` [Function]

Make a copy of *src* in its current state and store to *dest*.

This intrinsic is the same as the GCC built-in function `__builtin_va_copy`.

`void core.stdc.stdarg.va_end (va_list ap)` [Function]

Destroy *ap* so that it is no longer usable.

This intrinsic is the same as the GCC built-in function `__builtin_va_end`.

`void core.stdc.stdarg.va_start (T)(out va_list ap, ref T parmn)` [Template]

Initialize *ap* so that it can be used to access the variable arguments that follow the named argument *parmn*.

This intrinsic is the same as the GCC built-in function `__builtin_va_start`.

2.5.5 Volatile Intrinsics

The following functions are a collection of intrinsics for volatile operations, available by importing the `core.volatile` module.

Calls to them are guaranteed to not be removed (as dead assignment elimination or presumed to have no effect) or reordered in the same thread.

These reordering guarantees are only made with regards to other operations done through these functions; the compiler is free to reorder regular loads/stores with regards to loads/stores done through these functions.

This is useful when dealing with memory-mapped I/O (MMIO) where a store can have an effect other than just writing a value, or where sequential loads with no intervening stores can retrieve different values from the same location due to external stores to the location.

These functions will, when possible, do the load/store as a single operation. In general, this is possible when the size of the operation is less than or equal to `(void*).sizeof`, although some targets may support larger operations. If the load/store cannot be done as a single operation, multiple smaller operations will be used.

These are not to be conflated with atomic operations. They do not guarantee any atomicity. This may be provided by coincidence as a result of the instructions used on the target, but this should not be relied on for portable programs. Further, no memory fences are implied by these functions. They should not be used for communication between threads. They may be used to guarantee a write or read cycle occurs at a specified address.

`ubyte core.volatile.volatileLoad (ubyte* ptr)` [Function]
`ushort core.volatile.volatileLoad (ushort* ptr)` [Function]
`uint core.volatile.volatileLoad (uint* ptr)` [Function]
`ulong core.volatile.volatileLoad (ulong* ptr)` [Function]

Read value from the memory location indicated by *ptr*.

`ubyte core.volatile.volatileStore (ubyte* ptr, ubyte value)` [Function]
`ushort core.volatile.volatileStore (ushort* ptr, ushort value)` [Function]
`uint core.volatile.volatileStore (uint* ptr, uint value)` [Function]
`ulong core.volatile.volatileStore (ulong* ptr, ulong value)` [Function]

Write *value* to the memory location indicated by *ptr*.

2.5.6 CTFE Intrinsics

The following functions are only treated as intrinsics during compile-time function execution (CTFE) phase of compilation to allow more functions to be computable at compile-time, either because their generic implementations are too complex, or do some low-level bit manipulation of floating point types.

Calls to these functions that exist after CTFE has finished will get standard code-generation without any special compiler intrinsic support.

`float std.math.exponential.exp (float x)` [Function]
`double std.math.exponential.exp (double x)` [Function]
`real std.math.exponential.exp (real x)` [Function]

Calculates e^x.

This function is evaluated during CTFE as the GCC built-in function `__builtin_exp`.

`float std.math.exponential.expm1 (float x)` [Function]
`double std.math.exponential.expm1 (double x)` [Function]
`real std.math.exponential.expm1 (real x)` [Function]

Calculates $e^x - 1.0$.

This function is evaluated during CTFE as the GCC built-in function `__builtin_expm1`.

`float std.math.exponential.exp2 (float x)` [Function]
`double std.math.exponential.exp2 (double x)` [Function]
`real std.math.exponential.exp2 (real x)` [Function]

Calculates 2^x.

This function is evaluated during CTFE as the GCC built-in function `__builtin_exp2`.

`float std.math.exponential.log (float x)` [Function]
`double std.math.exponential.log (double x)` [Function]
`real std.math.exponential.log (real x)` [Function]

Calculate the natural logarithm of x.

This function is evaluated during CTFE as the GCC built-in function `__builtin_log`.

`float std.math.exponential.log10 (float x)` [Function]
`double std.math.exponential.log10 (double x)` [Function]
`real std.math.exponential.log10 (real x)` [Function]

Calculates the base-10 logarithm of x.

This function is evaluated during CTFE as the GCC built-in function `__builtin_log10`.

`float std.math.exponential.log2 (float x)` [Function]
`double std.math.exponential.log2 (double x)` [Function]
`real std.math.exponential.log2 (real x)` [Function]

Calculates the base-2 logarithm of x.

This function is evaluated during CTFE as the GCC built-in function `__builtin_log2`.

`Largest!(F, G) std.math.exponential.pow (F, G) (F x, G y)` [Template]
`real std.math.exponential.pow (I, F)(I x, F y)` [Template]

Calculates x^y, where y is a float.

This function is evaluated during CTFE as the GCC built-in function `__builtin_pow`.

`F std.math.exponential.pow (F, G) (F x, G n)` [Template]

Calculates x^n, where n is an integer.

This function is evaluated during CTFE as the GCC built-in function `__builtin_powi`.

`real std.math.operations.fma (real x, real y, real z)` [Function]

Returns `(x * y) + z`, rounding only once according to the current rounding mode.

This function is evaluated during CTFE as the GCC built-in function `__builtin_fma`.

`F std.math.operations.fmax (F)(const F x, const F y)` [Template]

Returns the larger of x and y.

This function is evaluated during CTFE as the GCC built-in function `__builtin_fmax`.

`F std.math.operations.fmin (F)(const F x, const F y)` [Template]

Returns the smaller of x and y.

This function is evaluated during CTFE as the GCC built-in function `__builtin_fmin`.

`float std.math.rounding.ceil (float x)` [Function]
`double std.math.rounding.ceil (double x)` [Function]
`real std.math.rounding.ceil (real x)` [Function]

Returns the value of x rounded upward to the next integer (toward positive infinity).

This function is evaluated during CTFE as the GCC built-in function `__builtin_ceil`.

`float std.math.rounding.floor (float x)` [Function]
`double std.math.rounding.floor (double x)` [Function]
`real std.math.rounding.floor (real x)` [Function]

Returns the value of *x* rounded downward to the next integer (toward negative infinity).

This function is evaluated during CTFE as the GCC built-in function `__builtin_floor`.

`real std.math.rounding.round (real x)` [Function]

Return the value of *x* rounded to the nearest integer. If the fractional part of *x* is exactly 0.5, the return value is rounded away from zero.

This function is evaluated during CTFE as the GCC built-in function `__builtin_round`.

`real std.math.rounding.trunc (real x)` [Function]

Returns the integer portion of *x*, dropping the fractional portion.

This function is evaluated during CTFE as the GCC built-in function `__builtin_trunc`.

`R std.math.traits.copysign (R, X)(R to, X from)` [Template]

Returns a value composed of *to* with *from*'s sign bit.

This function is evaluated during CTFE as the GCC built-in function `__builtin_copysign`.

`bool std.math.traits.isFinite (X)(X x)` [Template]

Returns true if *x* is finite.

This function is evaluated during CTFE as the GCC built-in function `__builtin_isfinite`.

`bool std.math.traits.isInfinity (X)(X x)` [Template]

Returns true if *x* is infinite.

This function is evaluated during CTFE as the GCC built-in function `__builtin_isinf`.

`bool std.math.traits.isNaN (X)(X x)` [Template]

Returns true if *x* is NaN.

This function is evaluated during CTFE as the GCC built-in function `__builtin_isnan`.

`float std.math.trigoometry.tan (float x)` [Function]
`double std.math.trigoometry.tan (double x)` [Function]
`real std.math.trigonometry.tan (real x)` [Function]

Returns tangent of *x*, where *x* is in radians.

This intrinsic is the same as the GCC built-in function `__builtin_tan`.

2.6 Predefined Pragmas

The `pragma` operator is used as a way to pass special information to the implementation and allow the addition of vendor specific extensions. The standard predefined pragmas are documented by the D language specification hosted at https://dlang.org/spec/pragma.html#predefined-pragmas. A D compiler must recognize, but is free to ignore any pragma in this list.

Where a pragma is ignored, the GNU D compiler will emit a warning when the `-Wunknown-pragmas` option is seen on the command-line.

`pragma(crt_constructor)`
: `pragma(crt_constructor)` annotates a function so it is run after the C runtime library is initialized and before the D runtime library is initialized. Functions with this pragma must return `void`.

```
pragma(crt_constructor) void init() { }
```

`pragma(crt_destructor)`
: `pragma(crt_destructor)` annotates a function so it is run after the D runtime library is terminated and before the C runtime library is terminated. Calling `exit` function also causes the annotated functions to run. Functions with this pragma must return `void`.

```
pragma(crt_destructor) void init() { }
```

`pragma(inline)`
`pragma(inline, false)`
`pragma(inline, true)`
: `pragma(inline)` affects whether functions are declared inlined or not. The pragma takes two forms. In the first form, inlining is controlled by the command-line options for inlining.

Functions annotated with `pragma(inline, false)` are marked uninlinable. Functions annotated with `pragma(inline, true)` are always inlined.

`pragma(lib)`
: This pragma is accepted, but has no effect.

```
pragma(lib, "advapi32");
```

`pragma(linkerDirective)`
: This pragma is accepted, but has no effect.

```
pragma(linkerDirective, "/FAILIFMISMATCH:_ITERATOR_DEBUG_LEVEL=2");
```

`pragma(mangle)`
: `pragma(mangle, "symbol_name")` overrides the default mangling for a function or variable symbol. The symbol name can be any expression that must evaluate at compile time to a string literal. This enables linking to a symbol which is a D keyword, since an identifier cannot be a keyword.

Targets are free to apply a prefix to the user label of the symbol name in assembly. For example, on `x86_64-apple-darwin`, ‘`symbol_name`’ would produce ‘`_symbol_name`’. If the mangle string begins with ‘`*`’, then `pragma(mangle)` will output the rest of the string unchanged.

```
pragma(mangle, "body")
```

```
extern(C) void body_func();

pragma(mangle, "function")
extern(C++) struct _function {}
```

`pragma(msg)`

: `pragma(msg, "message")` causes the compiler to print an informational message with the text '`message`'. The pragma accepts multiple arguments, each to which is evaluated at compile time and then all are combined into one concatenated message.

```
pragma(msg, "compiling...", 6, 1.0); // prints "compiling...61.0"
```

`pragma(printf)`
`pragma(scanf)`

: `pragma(printf)` and `pragma(scanf)` specifies that a function declaration with `printf` or `scanf` style arguments that should be type-checked against a format string.

A printf-like or scanf-like function can either be an `extern(C)` or `extern(C++)` function with a *format* parameter accepting a pointer to a 0-terminated `char` string, immediately followed by either a `...` variadic argument list or a parameter of type `va_list` as the last parameter.

```
extern(C):
pragma(printf)
int printf(scope const char* format, scope const ...);

pragma(scanf)
int vscanf(scope const char* format, va_list arg);
```

`pragma(startaddress)`

: This pragma is accepted, but has no effect.

```
void foo() { }
pragma(startaddress, foo);
```

2.7 Predefined Versions

Several conditional version identifiers are predefined; you use them without supplying their definitions. They fall into three classes: standard, common, and target-specific.

Predefined version identifiers from this list cannot be set from the command line or from version statements. This prevents things like both `Windows` and `linux` being simultaneously set.

2.7.1 Standard Predefined Versions

The standard predefined versions are documented by the D language specification hosted at `https://dlang.org/spec/version.html#predefined-versions`.

`all`
`none`

: Version `none` is never defined; used to just disable a section of code. Version `all` is always defined; used as the opposite of `none`.

`BigEndian`
`LittleEndian`
: These versions reflect the byte order of multi-byte data in memory. `LittleEndian` is set when the least significant byte is first. `BigEndian` is set when the most significant byte is first.

`CRuntime_Bionic`
`CRuntime_Glibc`
`CRuntime_Microsoft`
`CRuntime_Musl`
`CRuntime_Newlib`
`CRuntime_UClibc`
: These versions reflect which standard C library is being linked in. `CRuntime_Bionic` is set when Bionic is the default C library. `CRuntime_Glibc` is set when GLIBC is the default C library. `CRuntime_Microsoft` is set when MSVCRT is the default C library. `CRuntime_Musl` is set when musl is the default C library. `CRuntime_Newlib` is set when Newlib is the default C library. `CRuntime_UClibc` is set when uClibc is the default C library.

`CppRuntime_Gcc`
: This version is defined when the standard C++ library being linked in is `libstdc++`.

`D_BetterC`
: This version is defined when the standard D libraries are not being implicitly linked in. This also implies that features of the D language that rely on exceptions, module information, or run-time type information are disabled as well. Enabled by `-fno-druntime`.

`D_Coverage`
: This version is defined when code coverage analysis instrumentation is being generated. Enabled by `-ftest-coverage`.

`D_Ddoc`
: This version is defined when Ddoc documentation is being generated. Enabled by `-fdoc`.

`D_Exceptions`
: This version is defined when exception handling is supported. Disabled by `-fno-exceptions`.

`D_HardFloat`
`D_SoftFloat`
: These versions reflect the floating-point ABI in use by the target. `D_HardFloat` is set when the target hardware has a floating-point unit. `D_SoftFloat` is set when the target hardware does not have a floating-point unit.

`D_Invariants`
: This version is defined when checks are being emitted for class invariants and struct invariants. Enabled by `-finvariants`.

`D_LP64`
: This version is defined when pointers are 64-bits. Not to be confused with with C's `__LP64__` model.

`D_ModuleInfo`
: This version is defined when run-time module information (also known as `ModuleInfo`) is supported. Disabled by `-fno-moduleinfo`.

`D_NoBoundsChecks`
: This version is defined when array bounds checks are disabled. Enabled by `-fno-bounds-checks`.

`D_Optimized`
: This version is defined in all optimizing compilations.

`D_PIC`
: This version is defined when position-independent code is being generated. Enabled by `-fPIC`.

`D_PIE`
: This version is defined when position-independent code that can be only linked into executables is being generated. Enabled by `-fPIE`.

`D_PreConditions`
: This version is defined when checks are being emitted for `in` contracts. Disabled by `-fno-preconditions`.

`D_PostConditions`
: This version is defined when checks are being emitted for `out` contracts. Disabled by `-fno-postconditions`.

`D_TypeInfo`
: This version is defined when run-time type information (also known as `TypeInfo`) is supported. Disabled by `-fno-rtti`.

`D_Version2`
: This version defined when this is a D version 2 compiler.

`unittest`
: This version is defined when the `unittest` code is being compiled in. Enabled by `-funittest`.

2.7.2 Common Predefined Versions

The common predefined macros are GNU D extensions. They are available with the same meanings regardless of the machine or operating system on which you are using GNU D. Their names all start with `GNU`.

`GNU`
: This version is defined by the GNU D compiler. If all you need to know is whether or not your D program is being compiled by GDC, or a non-GDC compiler, you can simply test `version(GNU)`.

`GNU_CET`
: This version is defined when `-fcf-protection` is used. The protection level is also set in `__traits(getTargetInfo, "CET")` (see Section 2.9 [Traits], page 37).

`GNU_DWARF2_Exceptions`
`GNU_SEH_Exceptions`
`GNU_SjLj_Exceptions`
: These versions reflect the mechanism that will be used for exception handling by the target. `GNU_DWARF2_Exceptions` is defined when the target uses

DWARF 2 exceptions. `GNU_SEH_Exceptions` is defined when the target uses SEH exceptions. `GNU_SjLj_Exceptions` is defined when the target uses the `setjmp`/`longjmp`-based exception handling scheme.

`GNU_EMUTLS`
: This version is defined if the target does not support thread-local storage, and an emulation layer is used instead.

`GNU_InlineAsm`
: This version is defined when `asm` statements use GNU D style syntax. (see Section 2.4 [Inline Assembly], page 21)

`GNU_StackGrowsDown`
: This version is defined if pushing a word onto the stack moves the stack pointer to a smaller address, and is undefined otherwise.

2.7.3 Target-specific Predefined Versions

The D compiler normally predefines several versions that indicate what type of system and machine is in use. They are obviously different on each target supported by GCC.

`AArch64` Version relating to the AArch64 family of processors.

`Android` Version relating to the Android platform.

`ARM`
`ARM_HardFloat`
`ARM_SoftFloat`
`ARM_SoftFP`
`ARM_Thumb`
: Versions relating to the ARM family of processors.

`Cygwin` Version relating to the Cygwin environment.

`darwin` Deprecated; use `OSX` instead.

`DragonFlyBSD`
: Versions relating to DragonFlyBSD systems.

`FreeBSD`

`FreeBSD_9`
`FreeBSD_10`
`FreeBSD_11`
`FreeBSD_...`
: Versions relating to FreeBSD systems. The FreeBSD major version number is inferred from the target triplet.

`HPPA`
`HPPA64` Versions relating to the HPPA family of processors.

`Hurd` Version relating to GNU Hurd systems.

`linux` Version relating to Linux systems.

`MinGW` Version relating to the MinGW environment.

`MIPS32`
`MIPS64`
`MIPS_EABI`
`MIPS_HardFloat`
`MIPS_N32`
`MIPS_N64`
`MIPS_O32`
`MIPS_O64`
`MIPS_SoftFloat`
: Versions relating to the MIPS family of processors.

`NetBSD` Version relating to NetBSD systems.

`OpenBSD` Version relating to OpenBSD systems.

`OSX` Version relating to OSX systems.

`Posix` Version relating to POSIX systems (includes Linux, FreeBSD, OSX, Solaris, etc).

`PPC`
`PPC64`
`PPC_HardFloat`
`PPC_SoftFloat`
: Versions relating to the PowerPC family of processors.

`RISCV32`
`RISCV64` Versions relating to the RISC-V family of processors.

`S390`
`SystemZ` Versions relating to the S/390 and System Z family of processors.

`S390X` Deprecated; use `SystemZ` instead.

`Solaris` Versions relating to Solaris systems.

`SPARC`
`SPARC64`
`SPARC_HardFloat`
`SPARC_SoftFloat`
`SPARC_V8Plus`
: Versions relating to the SPARC family of processors.

`Thumb` Deprecated; use `ARM_Thumb` instead.

`D_X32`
`X86`
`X86_64` Versions relating to the x86-32 and x86-64 family of processors.

`Windows`
`Win32`
`Win64` Versions relating to Microsoft Windows systems.

2.8 Special Enums

Special `enum` names are used to represent types that do not have an equivalent basic D type. For example, C++ types used by the C++ name mangler.

Special enums are declared opaque, with a base type explicitly set. Unlike regular opaque enums, special enums can be used as any other value type. They have a default `.init` value, as well as other enum properties available (`.min`, `.max`). Special enums can be declared in any module, and will be recognized by the compiler.

```
import gcc.builtins;
enum __c_long : __builtin_clong;
__c_long var = 0x800A;
```

The following identifiers are recognized by GNU D.

`__c_complex_double`
: C `_Complex double` type.

`__c_complex_float`
: C `_Complex float` type.

`__c_complex_real`
: C `_Complex long double` type.

`__c_long`
: C++ `long` type.

`__c_longlong`
: C++ `long long` type.

`__c_long_double`
: C `long double` type.

`__c_ulong`
: C++ `unsigned long` type.

`__c_ulonglong`
: C++ `unsigned long long` type.

`__c_wchar_t`
: C++ `wchar_t` type.

The `core.stdc.config` module declares the following shorthand alias types for convenience: `c_complex_double`, `c_complex_float`, `c_complex_real`, `cpp_long`, `cpp_longlong`, `c_long_double`, `cpp_ulong`, `cpp_ulonglong`.

It may cause undefined behavior at runtime if a special enum is declared with a base type that has a different size to the target C/C++ type it is representing. The GNU D compiler will catch such declarations and emit a warning when the `-Wmismatched-special-enum` option is seen on the command-line.

2.9 Traits

Traits are extensions to the D programming language to enable programs, at compile time, to get at information internal to the compiler. This is also known as compile time reflection.

GNU D implements a `__traits(getTargetInfo)` trait that receives a string key as its argument. The result is an expression describing the requested target information.

```
version (OSX)
```

```
{
    static assert(__traits(getTargetInfo, "objectFormat") == "macho");
}
```

Keys for the trait are implementation defined, allowing target-specific data for exotic targets. A reliable subset exists which a D compiler must recognize. These are documented by the D language specification hosted at `https://dlang.org/spec/traits.html#getTargetInfo`.

The following keys are recognized by GNU D.

`CET` When `-fcf-protection` is used, the first bit is set to 1 for the value `branch` and the second bit is set to 1 for the value `return`.

`cppRuntimeLibrary`
The C++ runtime library affinity for this toolchain.

`cppStd` The version of the C++ standard supported by `extern(C++)` code, equivalent to the `__cplusplus` macro in a C++ compiler.

`floatAbi` Floating point ABI; may be '`hard`', '`soft`', or '`softfp`'.

`objectFormat`
Target object format.

2.10 Vector Extensions

CPUs often support specialized vector types and vector operations (aka media instructions). Vector types are a fixed array of floating or integer types, and vector operations operate simultaneously on them.

```
alias int4 = __vector(int[4]);
```

All the basic integer types can be used as base types, both as signed and as unsigned: `byte`, `short`, `int`, `long`. In addition, `float` and `double` can be used to build floating-point vector types, and `void` to build vectors of untyped data. Only sizes that are positive power-of-two multiples of the base type size are currently allowed.

The `core.simd` module has the following shorthand aliases for commonly supported vector types: `byte8`, `byte16`, `byte32`, `byte64`, `double1`, `double2`, `double4`, `double8`, `float2`, `float4`, `float8`, `float16`, `int2`, `int4`, `int8`, `int16`, `long1`, `long2`, `long4`, `long8`, `short4`, `short8`, `short16`, `short32`, `ubyte8`, `ubyte16`, `ubyte32`, `ubyte64`, `uint2`, `uint4`, `uint8`, `uint16`, `ulong1`, `ulong2`, `ulong4`, `ulong8`, `ushort4`, `ushort8`, `ushort16`, `ushort32`, `void8`, `void16`, `void32`, `void64`. All these aliases correspond to `__vector(type[N])`.

Which vector types are supported depends on the target. Only vector types that are implemented for the current architecture are supported at compile-time. Vector operations that are not supported in hardware cause GNU D to synthesize the instructions using a narrower mode.

```
alias v4i = __vector(int[4]);
alias v128f = __vector(float[128]);    // Error: not supported on this platform

int4 a, b, c;

c = a * b;    // Natively supported on x86 with SSE4
c = a / b;    // Always synthesized
```

Vector types can be used with a subset of normal D operations. Currently, GNU D allows using the following operators on these types: `+, -, *, /, unary+, unary-`.

```
alias int4 = __vector(int[4]);

int4 a, b, c;

c = a + b;
```

It is also possible to use shifting operators `<<`, `>>`, the modulus operator `%`, logical operations `&, |, ^`, and the complement operator `unary~` on integer-type vectors.

For convenience, it is allowed to use a binary vector operation where one operand is a scalar. In that case the compiler transforms the scalar operand into a vector where each element is the scalar from the operation. The transformation happens only if the scalar could be safely converted to the vector-element type. Consider the following code.

```
alias int4 = __vector(int[4]);

int4 a, b;
long l;

a = b + 1;    // a = b + [1,1,1,1];
a = 2 * b;    // a = [2,2,2,2] * b;

a = l + a;    // Error, incompatible types.
```

Vector comparison is supported with standard comparison operators: `==, !=, <, <=, >, >=`. Comparison operands can be vector expressions of integer-type or real-type. Comparison between integer-type vectors and real-type vectors are not supported. The result of the comparison is a vector of the same width and number of elements as the comparison operands with a signed integral element type.

Vectors are compared element-wise producing 0 when comparison is false and -1 (constant of the appropriate type where all bits are set) otherwise. Consider the following example.

```
alias int4 = __vector(int[4]);

int4 a = [1,2,3,4];
int4 b = [3,2,1,4];
int4 c;

c = a >  b;     // The result would be [0, 0,-1, 0]
c = a == b;     // The result would be [0,-1, 0,-1]
```

2.11 Vector Intrinsics

The following functions are a collection of vector operation intrinsics, available by importing the `gcc.simd` module.

`void gcc.simd.prefetch (bool` *`rw`*`, ubyte` *`locality`*`)` [Template]
    `(const(void)*` *`addr`*`)`

Emit the prefetch instruction. The value of *addr* is the address of the memory to prefetch. The value of *rw* is a compile-time constant one or zero; one means that the prefetch is preparing for a write to the memory address and zero, the default, means that the prefetch is preparing for a read. The value *locality* must be a compile-time constant integer between zero and three.

This intrinsic is the same as the GCC built-in function `__builtin_prefetch`.

```
for (i = 0; i < n; i++)
{
    import gcc.simd : prefetch;
    a[i] = a[i] + b[i];
    prefetch!(true, 1)(&a[i+j]);
    prefetch!(false, 1)(&b[i+j]);
    // ...
}
```

`V gcc.simd.loadUnaligned (V)(const V* p)` [Template]

Load unaligned vector from the address *p*.

```
float4 v;
ubyte[16] arr;

v = loadUnaligned(cast(float4*)arr.ptr);
```

`V gcc.simd.storeUnaligned (V)(V* p, V value)` [Template]

Store vector *value* to unaligned address *p*.

```
float4 v;
ubyte[16] arr;

storeUnaligned(cast(float4*)arr.ptr, v);
```

`V0 gcc.simd.shuffle (V0, V1, M)(V0 op1, V1 op2, M mask)` [Template]

`V gcc.simd.shuffle (V, M)(V op1, M mask)` [Template]

Construct a permutation of elements from one or two vectors, returning a vector of the same type as the input vector(s). The *mask* is an integral vector with the same width and element count as the output vector.

This intrinsic is the same as the GCC built-in function `__builtin_shuffle`.

```
int4 a = [1, 2, 3, 4];
int4 b = [5, 6, 7, 8];
int4 mask1 = [0, 1, 1, 3];
int4 mask2 = [0, 4, 2, 5];
int4 res;

res = shuffle(a, mask1);    // res is [1,2,2,4]
res = shuffle(a, b, mask2); // res is [1,5,3,6]
```

`V gcc.simd.shufflevector (V1, V2, M...)(V1 op1, V2 op2, M mask)` [Template]

`V gcc.simd.shufflevector (V, mask...)(V op1, V op2)` [Template]

Construct a permutation of elements from two vectors, returning a vector with the same element type as the input vector(s), and same length as the *mask*.

This intrinsic is the same as the GCC built-in function `__builtin_shufflevector`.

```
int8 a = [1, -2, 3, -4, 5, -6, 7, -8];
int4 b = shufflevector(a, a, 0, 2, 4, 6);   // b is [1,3,5,7]
int4 c = [-2, -4, -6, -8];
int8 d = shufflevector!(int8, 4, 0, 5, 1, 6, 2, 7, 3)(c, b); // d is a
```

`E gcc.simd.extractelement (V, int idx)(V val)` [Template]

Extracts a single scalar element from a vector *val* at a specified index *idx*.

```
int4 a = [0, 10, 20, 30];
int k = extractelement!(int4, 2)(a);    // a is 20
```

V **gcc.simd.insertelement** (V, int idx)(V val, B *e*) [Template]

Inserts a scalar element *e* into a vector *val* at a specified index *idx*.

```
int4 a = [0, 10, 20, 30];
int4 b = insertelement!(int4, 2)(a, 50); // b is [0,10,50,30]
```

V **gcc.simd.convertvector** (V, T)(T val) [Template]

Convert a vector *val* from one integral or floating vector type to another. The result is an integral or floating vector that has had every element cast to the element type of the return type.

This intrinsic is the same as the GCC built-in function `__builtin_convertvector`.

```
int4 a = [1, -2, 3, -4];
float4 b = [1.5, -2.5, 3, 7];
float4 c = convertvector!float4(a);    // c is [1,-2,3,-4]
double4 d = convertvector!double4(a);  // d is [1,-2,3,-4]
double4 e = convertvector!double4(b);  // e is [1.5,-2.5,3,7]
int4 f = convertvector!int4(b);        // f is [1,-2,3,7]
```

V0 **gcc.simd.blendvector** (V0, V1, M)(V0 *op0*, V1 *op1*, M *mask*) [Template]

Construct a conditional merge of elements from two vectors, returning a vector of the same type as the input vector(s). The *mask* is an integral vector with the same width and element count as the output vector.

```
int4 a = [1, 2, 3, 4];
int4 b = [3, 2, 1, 4];
auto c = blendvector(a, b, a > b);  // c is [3,2,3,4]
auto d = blendvector(a, b, a < b);  // d is [1,2,1,4]
```

2.12 Missing Features and Deviations

Some parts of the D specification are hard or impossible to implement with GCC, they should be listed here.

Bit Operation Intrinsics

The Digital Mars D compiler implements the `core.bitop` intrinsics `inp`, `inpw`, `inpl`, `outp`, `outpw`, and `outpl`. These are not recognized by GNU D. On most targets, equivalent intrinsics that have the same effect would be `core.volatile.loadVolatile` and `core.volatile.storeVolatile` respectively (see Section 2.5.5 [Volatile Intrinsics], page 27).

On x86 targets, if an `in` or `out` instruction is specifically required, that can be achieved using assembler statements instead.

```
ubyte inp(uint port)
{
    ubyte value;
    asm { "inb %w1, %b0" : "=a" (value) : "Nd" (port); }
    return value;
}

void outp(uint port, ushort value)
{
    asm { "outb %b0, %w1" : : "a" (value), "Nd" (port); }
}
```

Floating-Point Intermediate Values

GNU D uses a software compile-time floating-point type that assists in cross-compilation and support for arbitrary target `real` precisions wider than 80 bits. Because of this, the result of floating-point CTFE operations may have different results in GNU D compared with other D compilers that use the host's native floating-point type for storage and CTFE. In particular, GNU D won't overflow or underflow when a target real features a higher precision than the host. Differences also extend to `.stringof` representations of intermediate values due to formatting differences with `sprintf("%Lg")`.

```
version(GNU)
    assert((25.5).stringof ~ (3.01).stringof == "2.55e+13.01e+0");
else
    assert((25.5).stringof ~ (3.01).stringof == "25.53.01");
```

Function Calling Conventions

GNU D does not implement the `extern(D)` calling convention for x86 as described in the D specification hosted at `https://dlang.org/spec/abi.html#function_calling_conventions`.

Instead, there is no distinction between `extern(C)` and `extern(D)` other than name mangling.

ImportC Limitations

GNU D does not run the preprocessor automatically for any ImportC sources. Instead all C files are expected to be manually preprocessed before they are imported into the compilation.

Inline Assembler

GNU D does not implement the D inline assembler for x86 and x86_64 as described in the D specification hosted at `https://dlang.org/spec/iasm.html`. Nor does GNU D predefine the `D_InlineAsm_X86` and `D_InlineAsm_X86_64` version identifiers to indicate support.

The GNU D compiler uses an alternative, GCC-based syntax for inline assembler (see Section 2.4 [Inline Assembly], page 21).

Interfacing to Objective-C

GNU D does not support interfacing with Objective-C, nor its protocols, classes, subclasses, instance variables, instance methods and class methods. The `extern(Objective-C)` linkage is ignored, as are the `@optional` and `@selector` attributes. The `D_ObjectiveC` version identifier is not predefined for compilations.

Pragma Directives

Pragmas that are designed to embed information into object files or otherwise pass options to the linker are not supported by GNU D. These include `pragma(lib)`, `pragma(linkerDirective)`, and `pragma(startaddress)`.

SIMD Intrinsics

The Digital Mars D compiler implements the `core.simd` intrinsics `__simd`, `__simd_ib`, `__simd_sto`. These are not recognized by GNU D, nor does GNU D predefine the `D_SIMD` version identifier to indicate support.

On x86 targets, all intrinsics are available as functions in the `gcc.builtins` module, and have predictable equivalents.

```
version (DigitalMars)
{
    __simd(XMM.PSLLW, op1, op2);
    __simd_ib(XMM.PSLLW, op1, imm8);
}
version (GNU)
{
    __builtin_ia32_psllw(op1, op2);
    __builtin_ia32_psllwi(op1, imm8);
}
```

TypeInfo-based va_arg

The Digital Mars D compiler implements a version of `core.vararg.va_arg` that accepts a run-time `TypeInfo` argument for use when the static type is not known. This function is not implemented by GNU D. It is more portable to use variadic template functions instead.

GNU General Public License

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users’ Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see https://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `https://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `https://www.gnu.org/licenses/why-not-lgpl.html`.

GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Option Index

gdc's command line options are indexed here without any initial '-' or '--'. Where an option has both positive and negative forms (such as -f*option* and -fno-*option*), relevant entries in the manual are indexed under the most appropriate form; it may sometimes be useful to look up both forms.

N

S

V

W

X

Keyword Index

M

N

O

P

R

S

T

U

V

W