GNU Fortran Internals

For GCC version 16.0.0 (pre-release)

(GCC)

The gfortran team

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

Short Contents

Table of Contents

1 Introduction

This manual documents the internals of `gfortran`, the GNU Fortran compiler.

Warning: This document, and the compiler it describes, are still under development. While efforts are made to keep it up-to-date, it might not accurately reflect the status of the most recent GNU Fortran compiler.

At present, this manual is very much a work in progress, containing miscellaneous notes about the internals of the compiler. It is hoped that at some point in the future it will become a reasonably complete guide; in the interim, GNU Fortran developers are strongly encouraged to contribute to it as a way of keeping notes while working on the compiler.

2 Code that Interacts with the User

2.1 Command-Line Options

Command-line options for `gfortran` involve four interrelated pieces within the Fortran compiler code.

The relevant command-line flag is defined in `lang.opt`, according to the documentation in Section “Options” in *GNU Compiler Collection Internals*. This is then processed by the overall GCC machinery to create the code that enables `gfortran` and `gcc` to recognize the option in the command-line arguments and call the relevant handler function.

This generated code calls the `gfc_handle_option` code in `options.cc` with an enumerator variable indicating which option is to be processed, and the relevant integer or string values associated with that option flag. Typically, `gfc_handle_option` uses these arguments to set global flags which record the option states.

The global flags that record the option states are stored in the `gfc_option_t` struct, which is defined in `gfortran.h`. Before the options are processed, initial values for these flags are set in `gfc_init_option` in `options.cc`; these become the default values for the options.

2.2 Error Handling

The GNU Fortran compiler's parser operates by testing each piece of source code against a variety of matchers. In some cases, if these matchers do not match the source code, they will store an error message in a buffer. If the parser later finds a matcher that does correctly match the source code, then the buffered error is discarded. However, if the parser cannot find a match, then the buffered error message is reported to the user. This enables the compiler to provide more meaningful error messages even in the many cases where (erroneous) Fortran syntax is ambiguous due to things like the absence of reserved keywords.

As an example of how this works, consider the following line:

```
IF = 3
```

Hypothetically, this may get passed to the matcher for an `IF` statement. Since this could plausibly be an erroneous `IF` statement, the matcher will buffer an error message reporting the absence of an expected ‘`(`’ following an `IF`. Since no matchers reported an error-free match, however, the parser will also try matching this against a variable assignment. When `IF` is a valid variable, this will be parsed as an assignment statement, and the error discarded. However, when `IF` is not a valid variable, this buffered error message will be reported to the user.

The error handling code is implemented in `error.cc`. Errors are normally entered into the buffer with the `gfc_error` function. Warnings go through a similar buffering process, and are entered into the buffer with `gfc_warning`. There is also a special-purpose function, `gfc_notify_std`, for things which have an error/warning status that depends on the currently-selected language standard.

The `gfc_error_check` function checks the buffer for errors, reports the error message to the user if one exists, clears the buffer, and returns a flag to the user indicating whether or not an error existed. To check the state of the buffer without changing its state or reporting the errors, the `gfc_error_flag_test` function can be used. The `gfc_clear_error` function will clear out any errors in the buffer, without reporting them. The `gfc_warning_check` and `gfc_clear_warning` functions provide equivalent functionality for the warning buffer.

Only one error and one warning can be in the buffers at a time, and buffering another will overwrite the existing one. In cases where one may wish to work on a smaller piece of source code without disturbing an existing error state, the `gfc_push_error`, `gfc_pop_error`, and `gfc_free_error` mechanism exists to implement a stack for the error buffer.

For cases where an error or warning should be reported immediately rather than buffered, the `gfc_error_now` and `gfc_warning_now` functions can be used. Normally, the compiler will continue attempting to parse the program after an error has occurred, but if this is not appropriate, the `gfc_fatal_error` function should be used instead. For errors that are always the result of a bug somewhere in the compiler, the `gfc_internal_error` function should be used.

The syntax for the strings used to produce the error/warning message in the various error and warning functions is similar to the `printf` syntax, with '`%`'-escapes to insert variable values. The details, and the allowable codes, are documented in the `error_print` function in `error.cc`.

3 Frontend Data Structures

This chapter should describe the details necessary to understand how the various `gfc_*` data are used and interact. In general it is advisable to read the code in `dump-parse-tree.cc` as its routines should exhaust all possible valid combinations of content for these structures.

3.1 gfc_code

The executable statements in a program unit are represented by a nested chain of `gfc_code` structures. The type of statement is identified by the `op` member of the structure, the different possible values are enumerated in `gfc_exec_op`. A special member of this `enum` is `EXEC_NOP` which is used to represent the various `END` statements if they carry a label. Depending on the type of statement some of the other fields will be filled in. Fields that are generally applicable are the `next` and `here` fields. The former points to the next statement in the current block or is `NULL` if the current statement is the last in a block, `here` points to the statement label of the current statement.

If the current statement is one of `IF`, `DO`, `SELECT` it starts a block, i.e. a nested level in the program. In order to represent this, the `block` member is set to point to a `gfc_code` structure whose `next` member starts the chain of statements inside the block; this structure's `op` member should be set to the same value as the parent structure's `op` member. The `SELECT` and `IF` statements may contain various blocks (the chain of `ELSE IF` and `ELSE` blocks or the various `CASE`s, respectively). These chains are linked-lists formed by the `block` members.

Consider the following example code:

```
IF (foo < 20) THEN
  PRINT *, "Too small"
  foo = 20
ELSEIF (foo > 50) THEN
  PRINT *, "Too large"
  foo = 50
ELSE
  PRINT *, "Good"
END IF
```

This statement-block will be represented in the internal gfortran tree as follows, were the horizontal link-chains are those induced by the `next` members and vertical links down are those of `block`. '`==|`' and '`--|`' mean `NULL` pointers to mark the end of a chain:

```
... ==> IF ==> ...
        |
        +--> IF foo < 20 ==> PRINT *, "Too small" ==> foo = 20 ==|
             |
             +--> IF foo > 50 ==> PRINT *, "Too large" ==> foo = 50 ==|
                  |
                  +--> ELSE ==> PRINT *, "Good" ==|
                       |
                       +--|
```

3.1.1 IF Blocks

Conditionals are represented by `gfc_code` structures with their `op` member set to `EXEC_IF`. This structure's `block` member must point to another `gfc_code` node that is the header of the if-block. This header's `op` member must be set to `EXEC_IF`, too, its `expr` member holds the condition to check for, and its `next` should point to the code-chain of the statements to execute if the condition is true.

If in addition an `ELSEIF` or `ELSE` block is present, the `block` member of the if-block-header node points to yet another `gfc_code` structure that is the header of the elseif- or else-block. Its structure is identical to that of the if-block-header, except that in case of an `ELSE` block without a new condition the `expr` member should be `NULL`. This block can itself have its `block` member point to the next `ELSEIF` or `ELSE` block if there's a chain of them.

3.1.2 Loops

`DO` loops are stored in the tree as `gfc_code` nodes with their `op` set to `EXEC_DO` for a `DO` loop with iterator variable and to `EXEC_DO_WHILE` for infinite `DO`s and `DO WHILE` blocks. Their `block` member should point to a `gfc_code` structure heading the code-chain of the loop body; its `op` member should be set to `EXEC_DO` or `EXEC_DO_WHILE`, too, respectively.

For `DO WHILE` loops, the loop condition is stored on the top `gfc_code` structure's `expr` member; `DO` forever loops are simply `DO WHILE` loops with a constant `.TRUE.` loop condition in the internal representation.

Similarly, `DO` loops with an iterator have instead of the condition their `ext.iterator` member set to the correct values for the loop iterator variable and its range.

3.1.3 `SELECT` Statements

A `SELECT` block is introduced by a `gfc_code` structure with an `op` member of `EXEC_SELECT` and `expr` containing the expression to evaluate and test. Its `block` member starts a list of `gfc_code` structures linked together by their `block` members that stores the various `CASE` parts.

Each `CASE` node has its `op` member set to `EXEC_SELECT`, too, its `next` member points to the code-chain to be executed in the current case-block, and `extx.case_list` contains the case-values this block corresponds to. The `block` member links to the next case in the list.

3.1.4 `BLOCK` and `ASSOCIATE`

The code related to a `BLOCK` statement is stored inside an `gfc_code` structure (say *c*) with `c.op` set to `EXEC_BLOCK`. The `gfc_namespace` holding the locally defined variables of the `BLOCK` is stored in `c.ext.block.ns`. The code inside the construct is in `c.code`.

`ASSOCIATE` constructs are based on `BLOCK` and thus also have the internal storage structure described above (including `EXEC_BLOCK`). However, for them `c.ext.block.assoc` is set additionally and points to a linked list of `gfc_association_list` structures. Those structures basically store a link of associate-names to target expressions. The associate-names themselves are still also added to the `BLOCK`'s namespace as ordinary symbols, but they have their `gfc_symbol`'s member `assoc` set also pointing to the association-list structure. This way associate-names can be distinguished from ordinary variables and their target expressions identified.

For association to expressions (as opposed to variables), at the very beginning of the BLOCK construct assignments are automatically generated to set the corresponding variables to their target expressions' values, and later on the compiler simply disallows using such associate-names in contexts that may change the value.

3.2 gfc_expr

Expressions and "values", including constants, variable-, array- and component-references as well as complex expressions consisting of operators and function calls are internally represented as one or a whole tree of gfc_expr objects. The member expr_type specifies the overall type of an expression (for instance, EXPR_CONSTANT for constants or EXPR_VARIABLE for variable references). The members ts and rank as well as shape, which can be NULL, specify the type, rank and, if applicable, shape of the whole expression or expression tree of which the current structure is the root. where is the locus of this expression in the source code.

Depending on the flavor of the expression being described by the object (that is, the value of its expr_type member), the corresponding structure in the value union will usually contain additional data describing the expression's value in a type-specific manner. The ref member is used to build chains of (array-, component- and substring-) references if the expression in question contains such references, see below for details.

3.2.1 Constants

Scalar constants are represented by gfc_expr nodes with their expr_type set to EXPR_CONSTANT. The constant's value shall already be known at compile-time and is stored in the logical, integer, real, complex or character struct inside value, depending on the constant's type specification.

3.2.2 Operators

Operator-expressions are expressions that are the result of the execution of some operator on one or two operands. The expressions have an expr_type of EXPR_OP. Their value.op structure contains additional data.

op1 and optionally op2 if the operator is binary point to the two operands, and operator or uop describe the operator that should be evaluated on these operands, where uop describes a user-defined operator.

3.2.3 Function Calls

If the expression is the return value of a function-call, its expr_type is set to EXPR_FUNCTION, and symtree must point to the symtree identifying the function to be called. value.function.actual holds the actual arguments given to the function as a linked list of gfc_actual_arglist nodes.

The other members of value.function describe the function being called in more detail, containing a link to the intrinsic symbol or user-defined function symbol if the call is to an intrinsic or external function, respectively. These values are determined during resolution-phase from the structure's symtree member.

A special case of function calls are "component calls" to type-bound procedures; those have the expr_type EXPR_COMPCALL with value.compcall containing the argument list and the procedure called, while symtree and ref describe the object on which the procedure was called in the same way as a EXPR_VARIABLE expression would. See Section 4.1 [Type-bound Procedures], page 11.

3.2.4 Array- and Structure-Constructors

Array- and structure-constructors (one could probably call them "array-" and "derived-type constants") are gfc_expr structures with their expr_type member set to EXPR_ARRAY or EXPR_STRUCTURE, respectively. For structure constructors, symtree points to the derived-type symbol for the type being constructed.

The values for initializing each array element or structure component are stored as linked-list of gfc_constructor nodes in the value.constructor member.

3.2.5 Null

NULL is a special value for pointers; it can be of different base types. Such a NULL value is represented in the internal tree by a gfc_expr node with expr_type EXPR_NULL. If the base type of the NULL expression is known, it is stored in ts (that's for instance the case for default-initializers of ALLOCATABLE components), but this member can also be set to BT_UNKNOWN if the information is not available (for instance, when the expression is a pointer-initializer NULL()).

3.2.6 Variables and Reference Expressions

Variable references are gfc_expr structures with their expr_type set to EXPR_VARIABLE; their symtree should point to the variable that is referenced.

For this type of expression, it's also possible to chain array-, component- or substring-references to the original expression to get something like 'struct%component(2:5)', where component is either an array or a CHARACTER member of struct that is of some derived-type. Such a chain of references is achieved by a linked list headed by ref of the gfc_expr node. For the example above it would be ('==|' is the last NULL pointer):

```
EXPR_VARIABLE(struct) ==> REF_COMPONENT(component) ==> REF_ARRAY(2:5) ==|
```

If component is a string rather than an array, the last element would be a REF_SUBSTRING reference, of course. If the variable itself or some component referenced is an array and the expression should reference the whole array rather than being followed by an array-element or -section reference, a REF_ARRAY reference must be built as the last element in the chain with an array-reference type of AR_FULL. Consider this example code:

```
TYPE :: mytype
  INTEGER :: array(42)
END TYPE mytype

TYPE(mytype) :: variable
INTEGER :: local_array(5)

CALL do_something (variable%array, local_array)
```

The gfc_expr nodes representing the arguments to the 'do_something' call will have a reference-chain like this:

```
EXPR_VARIABLE(variable) ==> REF_COMPONENT(array) ==> REF_ARRAY(FULL) ==|
EXPR_VARIABLE(local_array) ==> REF_ARRAY(FULL) ==|
```

3.2.7 Constant Substring References

`EXPR_SUBSTRING` is a special type of expression that encodes a substring reference of a constant string, as in the following code snippet:

```
x = "abcde"(1:2)
```

In this case, `value.character` contains the full string's data as if it was a string constant, but the `ref` member is also set and points to a substring reference as described in the subsection above.

4 Internals of Fortran 2003 OOP Features

4.1 Type-bound Procedures

Type-bound procedures are stored in the `tb_sym_root` of the namespace `f2k_derived` associated with the derived-type symbol as `gfc_symtree` nodes. The name and symbol of these symtrees corresponds to the binding-name of the procedure, i.e. the name that is used to call it from the context of an object of the derived-type.

In addition, this type of symtrees stores in `n.tb` a struct of type `gfc_typebound_proc` containing the additional data needed: The binding attributes (like `PASS` and `NOPASS`, `NON_OVERRIDABLE` or the access-specifier), the binding's target(s) and, if the current binding overrides or extends an inherited binding of the same name, `overridden` points to this binding's `gfc_typebound_proc` structure.

4.1.1 Specific Bindings

For specific bindings (declared with `PROCEDURE`), if they have a passed-object argument, the passed-object dummy argument is first saved by its name, and later during resolution phase the corresponding argument is looked for and its position remembered as `pass_arg_num` in `gfc_typebound_proc`. The binding's target procedure is pointed-to by `u.specific`.

`DEFERRED` bindings are just like ordinary specific bindings, except that their `deferred` flag is set of course and that `u.specific` points to their "interface" defining symbol (might be an abstract interface) instead of the target procedure.

At the moment, all type-bound procedure calls are statically dispatched and transformed into ordinary procedure calls at resolution time; their actual argument list is updated to include at the right position the passed-object argument, if applicable, and then a simple procedure call to the binding's target procedure is built. To handle dynamic dispatch in the future, this will be extended to allow special code generation during the trans-phase to dispatch based on the object's dynamic type.

4.1.2 Generic Bindings

Bindings declared as `GENERIC` store the specific bindings they target as a linked list using nodes of type `gfc_tbp_generic` in `u.generic`. For each specific target, the parser records its symtree and during resolution this symtree is bound to the corresponding `gfc_typebound_proc` structure of the specific target.

Calls to generic bindings are handled entirely in the resolution-phase, where for the actual argument list present the matching specific binding is found and the call's target procedure (`value.compcall.tbp`) is re-pointed to the found specific binding and this call is subsequently handled by the logic for specific binding calls.

4.1.3 Calls to Type-bound Procedures

Calls to type-bound procedures are stored in the parse-tree as `gfc_expr` nodes of type `EXPR_COMPCALL`. Their `value.compcall.actual` saves the actual argument list of the call and `value.compcall.tbp` points to the `gfc_typebound_proc` structure of the binding to be called. The object in whose context the procedure was called is saved by combination of `symtree` and `ref`, as if the expression was of type `EXPR_VARIABLE`.

For code like this:

```
CALL myobj%procedure (arg1, arg2)
```

the `CALL` is represented in the parse-tree as a `gfc_code` node of type `EXEC_COMPCALL`. The `expr` member of this node holds an expression of type `EXPR_COMPCALL` of the same structure as mentioned above except that its target procedure is of course a `SUBROUTINE` and not a `FUNCTION`.

Expressions that are generated internally (as expansion of a type-bound operator call) may also use additional flags and members. `value.compcall.ignore_pass` signals that even though a `PASS` attribute may be present the actual argument list should not be updated because it already contains the passed-object. `value.compcall.base_object` overrides, if it is set, the base-object (that is normally stored in `symtree` and `ref` as mentioned above); this is needed because type-bound operators can be called on a base-object that need not be of type `EXPR_VARIABLE` and thus representable in this way. Finally, if `value.compcall.assign` is set, the call was produced in expansion of a type-bound assignment; this means that proper dependency-checking needs to be done when relevant.

4.2 Type-bound Operators

Type-bound operators are in fact basically just `GENERIC` procedure bindings and are represented much in the same way as those (see Section 4.1 [Type-bound Procedures], page 11).

They come in two flavours: User-defined operators (like `.MYOPERATOR.`) are stored in the `f2k_derived` namespace's `tb_uop_root` symtree exactly like ordinary type-bound procedures are stored in `tb_sym_root`; their symtrees' names are the operator-names (e.g. '`myoperator`' in the example). Intrinsic operators on the other hand are stored in the namespace's array member `tb_op` indexed by the intrinsic operator's enum value. Those need not be packed into `gfc_symtree` structures and are only `gfc_typebound_proc` instances.

When an operator call or assignment is found that cannot be handled in another way (i.e. neither matches an intrinsic nor interface operator definition) but that contains a derived-type expression, all type-bound operators defined on that derived-type are checked for a match with the operator call. If there's indeed a relevant definition, the operator call is replaced with an internally generated `GENERIC` type-bound procedure call to the respective definition and that call is further processed.

5 Generating the intermediate language for later stages.

This chapter deals with the transformation of gfortran's frontend data structures to the intermediate language used by the later stages of the compiler, the so-called middle end.

Data structures relating to this are found in the source files `trans*.h` and `trans-*.c`.

5.1 Basic data structures

Gfortran creates GENERIC as an intermediate language for the middle-end. Details about GENERIC can be found in the GCC manual.

The basic data structure of GENERIC is a `tree`. Everything in GENERIC is a `tree`, including types and statements. Fortunately for the gfortran programmer, `tree` variables are garbage-collected, so doing memory management for them is not necessary.

`tree` expressions are built using functions such as, for example, `fold_build2_loc`. For two tree variables `a` and `b`, both of which have the type `gfc_arry_index_type`, calculation `c = a * b` would be done by

```
c = fold_build2_loc (input_location, MULT_EXPR,
                     gfc_array_index_type, a, b);
```

The types have to agree, otherwise internal compiler errors will occur at a later stage. Expressions can be converted to a different type using `fold_convert`.

Accessing individual members in the `tree` structures should not be done. Rather, access should be done via macros.

One basic data structure is the `stmtblock_t` struct. This is used for holding a list of statements, expressed as `tree` expressions. If a block is created using `gfc_start_block`, it has its own scope for variables; if it is created using `gfc_init_block`, it does not have its own scope.

It is possible to

- Add an expression to the end of a block using `gfc_add_expr_to_block`
- Add an expression to the beginning of a block using `void gfc_prepend_expr_to_block`
- Make a block into a single `tree` using `gfc_finish_block`. For example, this is needed to put the contents of a block into the `if` or `else` branch of a `COND_EXPR`.

Variables are also `tree` expressions, they can be created using `gfc_create_var`. Assigning to a variable can be done with `gfc_add_modify`.

An example: Creating a default integer type variable in the current scope with the prefix "everything" in the `stmt_block` `block` and assigning the value 42 would be

```
tree var, *block;
/* Initialize block somewhere here.  */
var = gfc_create_var (integer_type_node, "everything");
gfc_add_modify (block, var, build_int_cst (integer_type_node, 42));
```

5.2 Converting Expressions to tree

Converting expressions to `tree` is done by functions called `gfc_conv_*`.

The central data structure for a GENERIC expression is the `gfc_se` structure. Its `expr` member is a `tree` that holds the value of the expression. A `gfc_se` structure is initialized using `gfc_init_se`; it needs to be embedded in an outer `gfc_se`.

Evaluating Fortran expressions often require things to be done before and after evaluation of the expression, for example code for the allocation of a temporary variable and its subsequent deallocation. Therefore, `gfc_se` contains the members `pre` and `post`, which point to `stmt_block` blocks for code that needs to be executed before and after evaluation of the expression.

When using a local `gfc_se` to convert some expression, it is often necessary to add the generated `pre` and `post` blocks to the `pre` or `post` blocks of the outer `gfc_se`. Code like this (lifted from `trans-expr.cc`) is fairly common:

```
gfc_se cont_se;
tree cont_var;

/* cont_var = is_contiguous (expr); .  */
gfc_init_se (&cont_se, parmse);
gfc_conv_is_contiguous_expr (&cont_se, expr);
gfc_add_block_to_block (&se->pre, &(&cont_se)->pre);
gfc_add_modify (&se->pre, cont_var, cont_se.expr);
gfc_add_block_to_block (&se->pre, &(&cont_se)->post);
```

Conversion functions which need a `gfc_se` structure will have a corresponding argument.

`gfc_se` also contains pointers to a `gfc_ss` and a `gfc_loopinfo` structure. These are needed by the scalarizer.

5.3 Translating statements

Translating statements to `tree` is done by functions called `gfc_trans_*`. These functions usually get passed a `gfc_code` structure, evaluate any expressions and then return a `tree` structure.

5.4 Accessing declarations

`gfc_symbol`, `gfc_charlen` and other front-end structures contain a `backend_decl` variable, which contains the `tree` used for accessing that entity in the middle-end.

Accessing declarations is usually done by functions called `gfc_get*`.

6 The LibGFortran Runtime Library

6.1 Symbol Versioning

In general, this capability exists only on a few platforms, thus there is a need for configure magic so that it is used only on those targets where it is supported.

The central concept in symbol versioning is the so-called map file, which specifies the version node(s) exported symbols are labeled with. Also, the map file is used to hide local symbols.

Some relevant references:

- GNU `ld` manual (`https://sourceware.org/binutils/docs/ld/VERSION.html`)
- ELF Symbol Versioning - Ulrich Depper (`https://www.akkadia.org/drepper/symbol-versioning`)
- How to Write Shared Libraries - Ulrich Drepper (see Chapter 3) (`https://www.akkadia.org/drepper/dsohowto.pdf`)

If one adds a new symbol to a library that should be exported, the new symbol should be mentioned in the map file and a new version node defined, e.g., if one adds a new symbols `foo` and `bar` to libgfortran for the next GCC release, the following should be added to the map file:

```
GFORTRAN_1.1 {
    global:
        foo;
        bar;
} GFORTRAN_1.0;
```

where `GFORTRAN_1.0` is the version node of the current release, and `GFORTRAN_1.1` is the version node of the next release where foo and bar are made available.

If one wants to change an existing interface, it is possible by using some asm trickery (from the `ld` manual referenced above):

```
__asm__(".symver original_foo,foo@");
__asm__(".symver old_foo,foo@VERS_1.1");
__asm__(".symver old_foo1,foo@VERS_1.2");
__asm__(".symver new_foo,foo@VERS_2.0");
```

In this example, `foo@` represents the symbol `foo` bound to the unspecified base version of the symbol. The source file that contains this example would define 4 C functions: `original_foo`, `old_foo`, `old_foo1`, and `new_foo`.

In this case the map file must contain `foo` in `VERS_1.1` and `VERS_1.2` as well as in `VERS_2.0`.

GNU Free Documentation License

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.

Index