# GNU C Compiler Internals

en.wikibooks.org

On the 28th of April 2012 the contents of the English as well as German Wikibooks and Wikipedia projects were licensed under Creative Commons Attribution-ShareAlike 3.0 Unported license. A URI to this license is given in the list of figures on page 63. If this document is a derived work from the contents of one of these projects and the content was still licensed by the project under this license at the time of derivation this document has to be licensed under the same, a similar or a compatible license, as stated in section 4b of the license. The list of contributors is included in chapter Contributors on page 61. The licenses GPL, LGPL and GFDL are included in chapter Licenses on page 67, since this book and/or parts of it may or may not be licensed under one or more of these licenses, and thus require inclusion of these licenses. The licenses of the figures are given in the list of figures on page 63. This PDF was generated by the LATEX typesetting software. The LATEX source code is included as an attachment (source.7z.txt) in this PDF file. To extract the source from the PDF file, you can use the pdfdetach tool including in the poppler suite, or the http://www. pdflabs.com/tools/pdftk-the-pdf-toolkit/ utility. Some PDF viewers may also let you save the attachment to a file. After extracting it from the PDF file you have to rename it to source.7z. To uncompress the resulting archive we recommend the use of http://www.7-zip.org/. The LATEX source itself was generated by a program written by Dirk Hünniger, which is freely available under an open source license from http://de.wikibooks.org/wiki/Benutzer:Dirk\_Huenniger/wb2pdf.

# Contents

1	GNU	J C Compiler Internals/Print version	3	
2	Intro 2.1 2.2 2.3	What is GCC?	<b>5</b> 5 6	
3	GNU	U C Compiler Architecture. Compiling an Expression	7	
4	Com 4.1 4.2	pilation of a function  Function Prolog and Epilog	21 21 22	
5	Fund 5.1 5.2	Ction calls Global Control-Flow Analysis	23 23 23	
6	Stac	kguard	27	
7	GEM Framework			
8	<b>C F</b> u 8.1	Inction Overloading Function overloading in C	<b>35</b> 35	
9	Retu 9.1	rn Address Defense Return Address Defense	<b>37</b> 37	
10	Add: 10.1 10.2 10.3 10.4 10.5 10.6 10.7	toString() method for each structure as in Java  Invoke a block of code from a function as in Ruby  Dereference function results when a structure is returned  Use functions to initialize a variable  Default values of function arguments as in C++  Reference parameters as in C++  GCC switches in object file  Type information at run-time	39 39 39 40 40 40 40	
11	11.1 11.2	Break up enormous functions	43 43 43 44	

	11.4 11.5	Delete garbage	44 44
12	Secu 12.1 12.2 12.3 12.4 12.5	Return address protection (RAD)	45 45 45 45 45 45
<b>13</b>	Link	S	47
14	Cont	ributors	49
15	15.1 15.2 15.3 15.4 15.5 15.6 15.7 15.8 15.9 15.10	7 Free Documentation License  1. APPLICABILITY AND DEFINITIONS  2. VERBATIM COPYING  3. COPYING IN QUANTITY  4. MODIFICATIONS  5. COMBINING DOCUMENTS  6. COLLECTIONS OF DOCUMENTS  7. AGGREGATION WITH INDEPENDENT WORKS  8. TRANSLATION  9. TERMINATION  10. FUTURE REVISIONS OF THIS LICENSE  11. RELICENSING	51 51 53 53 53 55 56 56 56 57
16	How	to use this License for your documents	59
17	Cont	ributors	61
Lis	st of I	Figures	63
18	Licer 18.1 18.2	GNU GENERAL PUBLIC LICENSE	67 68 69

# 1 GNU C Compiler Internals/Print version



This is the print version<sup>1</sup> of GNU C Compiler Internals<sup>2</sup>

You won't see this message or any elements not part of the book's content when you print or preview<sup>3</sup> this page.

#### GNU C Compiler Internals

The current, editable version of this book is available in Wikibooks, the open-content textbooks collection, at

- <sup>4</sup> Permission is granted to copy, distribute, and/or modify this document under the terms of the Creative Commons Attribution-ShareAlike 3.0 License<sup>5</sup>.
  - 1. Introduction
  - 2. GNU C Compiler Architecture. Compilation of an expression
  - 3. Compilation of a function
  - 4. Function cals
  - 5. Stackguard
  - 6. GEM Framework
  - 7. Function Overloading in C
  - 8. Return Address Defense
  - 9. Adding Syntactic Sugar
  - 10. Improving Code Style
  - 11. Security Enhancements
  - 12. Links
  - 13. Contributors

<sup>1</sup> http://en.wikibooks.org/wiki/Help:Print\_versions

<sup>2</sup> http://en.wikibooks.org/wiki/GNU\_C\_Compiler\_Internals

http://en.wikibooks.org//en.wikibooks.org/w/index.php?title=GNU\_C\_Compiler\_Internals/

Print\_version&action=purge&printable=yes

<sup>4</sup> http://en.wikibooks.org/wiki/GNU\_C\_Compiler\_Internals

http://en.wikibooks.org/wiki/Wikibooks:Creative\_Commons\_Attribution-ShareAlike\_3.0\_

 $<sup>^{\</sup>prime\prime}$  Unported\_License

# 2 Introduction

#### 2.1 What is GCC?

GNU Compiler Collection<sup>1</sup> (GCC) is a free software project that includes compilers for Ada<sup>2</sup>, C<sup>3</sup>, C++<sup>4</sup>, Fortran<sup>5</sup>, Java<sup>6</sup>, and Objective-C<sup>7</sup>, as well as libraries for these languages. It is capable of generating executables for a variety of platforms<sup>8</sup> including x86<sup>9</sup>, ARM<sup>10</sup>, MIPS<sup>11</sup>, PowerPC<sup>12</sup>, etc.

# 2.2 History of GCC

The homepage of GCC is <sup>13</sup>. The modern history of GCC begins with the GCC 2.95 release. This version was released on July 31, 1999. GCC 3.0 which is considered modern history for the C++ compiler was released on June 18, 2001. Additional branches were created later on. As of now, the active development branches are GCC 3.4 with a latest release on November 30, 2005, and GCC 4.0 released last time on September 28, 2005. GCC 4.1 was released on Feb 28 2006. GCC 4.2 is the development branch of GCC. The source code repository is available online<sup>14</sup>. GCC 4.2 (the trunk) is the only place where real development happens and new features can go. GCC has become popular in industry and academia. The availability of its source code allows one to add new features to the compiler. GCC is used in several source-code based security projects, that is, in the tools that instrument the source code of the program to make it more secure. However, very few documents describing the GCC internals have been published so far. When new functionality is implemented, the source code of GCC is modified directly. However, these compiler extensions are difficult to distribute because GCC is a really big program. A framework for creating modular extensions would greatly simplify the development of compiler extensions. GCC 4.0 and above includes SSA optimizers and should be used for further high level

<sup>1</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/GNU\_Compiler\_Collection

<sup>2</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/Ada\_programming\_language

<sup>3</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/C\_programming\_language

<sup>4</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/C%2B%2B

<sup>5</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/Fortran

<sup>6</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/Java\_programming\_language

<sup>7</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/Objective-C

<sup>8</sup> http://gcc.gnu.org/install/specific.html

<sup>9</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/x86\_architecture

<sup>10</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/ARM\_architecture

<sup>11</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/MIPS\_architecture

<sup>12</sup> http://en.wikibooks.org//en.wikipedia.org/wiki/PowerPC

<sup>13</sup> http://gcc.gnu.org

<sup>14</sup> http://gcc.gnu.org/viewcvs/branches/

optimizations and transformations. The RTL level should only be used for target specific optimizations and optimizations which are low level such as scheduling. Any GCC before 4.0 is in some minds an old dated compiler which was showing its age.

# 2.3 Purpose of this book

The purpose of this book is to address the demands of GCC hackers. We start with a description of GCC 3.4.1 architecture<sup>15</sup> focusing on the source code parser. We chose this version of GCC because we used this version mostly. Then we address the problem of extension development. We present the GCC Extensibility Modules<sup>16</sup> (GEM) project in the next chapter<sup>17</sup>. GEM provides a number of hooks throughout GCC source code. It is implemented as a patch to GCC. A GEM-based compiler extension is developed as a stand-alone program. When the extension is completed, only its source code is distributed compared with distributing the source code of the GCC if GEM is not used. We give examples that demonstrate GEM programming at the end of the book<sup>18</sup>.

Take home:

What this book is: a guide to GCC internals for extension programmers. GEM framework for developing modular extensions is introduced. What this book is not: a programming language reference, GCC installation manual.

 $<sup>15 \</sup>quad \hbox{\tt\#An\_Overview\_of\_GCC\_Architecture}$ 

<sup>16</sup> http://research.alexeysmirnov.name/gem

<sup>17 #</sup>GEM\_Framework

<sup>18 #</sup>Creating\_a\_Compiler\_Extension

# 3 GNU C Compiler Architecture. Compiling an Expression

#### 3.0.1 An Overview of GCC Architecture. Compilation of an expression.

This section is based on a Red Hat magazine article <sup>[1]¹</sup>. The GNU Compiler Collection (GCC) comprises a number of compilers for different programming languages. The main GCC executable **gcc** processes source files written in C, C++, Objective-C, Objective-C++, Java, Fortran, or Ada and produces an assembly file for each source file. It is a driver program that invokes the appropriate compilation programs depending on the language of the source file. For a C source file they are the preprocessor and compiler **cc1**, the assembler **as**, and the linker **collect2**. The first and the third programs come with a GCC distribution; the assembler is a part of the GNU binutils package. This book describes the internals of the preprocessor and compiler **cc1**. Each compiler includes three components: a front end, a middle end, and a back end. GCC compiles one file at a time. A source file goes through all three components one after another. Its representation is modified when it goes from a component to the next. Figure 1² illustrates the components and the source file representations associated with each of them. The abstract syntax tree (AST), register transfer language (RTL), and object are the main representations.



Figure 2 GCC front end, middle end, and back end with source file representations.

#### Main Representations

The purpose of the front end is to read the source file, parse it, and convert it into the standard abstract syntax tree (AST) representation. The AST is a dual-type representation: it is a tree where a node can have children and a list of statements where nodes are chained one after another. There is one front end for each programming language. The AST is

<sup>2</sup> http://en.wikibooks.org/wiki/File:GCC\_ast\_rtl\_obj.jpg

then used to generate a register-transfer language (RTL) tree. RTL is a hardware-based representation that corresponds to an abstract target architecture with an infinite number of registers. An RTL optimization pass optimizes the tree in the RTL form. Finally, a GCC back end generates the assembly code for the target architecture using the RTL representation. Examples of back ends are the x86 back end and the MIPS back end. In the next sections we describe the internals of the C front end and the x86 back end. The compiler starts with its initialization and command line options processing. After that the C front end preprocesses the source file, parses it and performs a number of optimizations. The back end then generates the assembly code for the target platform and saves it to a file.

#### 3.0.2 Auxiliary Data Structures

GCC has a number of additional data structures that facilitate code development, for example vector and heap. The macros defined in **vec.h** implement a set of templated vector types and associated interfaces. These templates are implemented with macros, as we're not in C++ land. The interface functions are typesafe and use static inline functions, sometimes backed by out-of-line generic functions. The vectors are designed to interoperate with the GTY<sup>3</sup> machinery. Because of the different behavior of structure objects, scalar objects and of pointers, there are three flavors, one for each of these variants. Both the pointer and structure object variants pass pointers to objects around -- in the former case the pointers are stored into the vector and in the latter case the pointers are dereferenced and the objects copied into the vector. The scalar object variant is suitable for int-like objects, and the vector elements are returned by value. There are both 'index' and 'iterate' accessors. The iterator returns a boolean iteration condition and updates the iteration variable passed by reference. Because the iterator will be inlined, the address-of can be optimized away. The vectors are implemented using the trailing array idiom, thus they are not resizeable without changing the address of the vector object itself. This means you cannot have variables or fields of vector type -- always use a pointer to a vector. The one exception is the final field of a structure, which could be a vector type. You will have to use the embedded size & embedded init calls to create such objects, and they will probably not be resizeable (so don't use the 'safe' allocation variants). The trailing array idiom is used (rather than a pointer to an array of data), because, if we allow NULL to also represent an empty vector, empty vectors occupy minimal space in the structure containing them. Each operation that increases the number of active elements is available in 'quick' and 'safe' variants. The former presumes that there is sufficient allocated space for the operation to succeed (it dies if there is not). The latter will reallocate the vector, if needed. Reallocation causes an exponential increase in vector size. If you know you will be adding N elements, it would be more efficient to use the reserve operation before adding the elements with the 'quick' operation. This will ensure there are at least as many elements as you ask for, it will exponentially increase if there are too few spare slots. If you want to reserve a specific number of slots, but do not want the exponential increase (for instance, you know this is the last allocation), use a negative number for reservation. You can also create a vector of a specific size from the get go. You should prefer the push and pop operations, as they append and remove from the end of the vector. If you need to remove several items in one go, use the truncate

<sup>3</sup> http://gcc.gnu.org/onlinedocs/gcc-4.3.4/gccint/GTY-Options.html

operation. The insert and remove operations allow you to change elements in the middle of the vector. There are two remove operations, one which preserves the element ordering 'ordered remove', and one which does not 'unordered remove'. The latter function copies the end element into the removed slot, rather than invoke a memmove operation. The 'lower bound' function will determine where to place an item in the array using insert that will maintain sorted order. When a vector type is defined, first a non-memory managed version is created. You can then define either or both garbage collected and heap allocated versions. The allocation mechanism is specified when the type is defined, and is therefore part of the type. If you need both gc'd and heap allocated versions, you still must have exactly one definition of the common non-memory managed base vector. If you need to directly manipulate a vector, then the 'address' accessor will return the address of the start of the vector. Also the 'space' predicate will tell you whether there is spare capacity in the vector. You will not normally need to use these two functions. Vector types are defined using a DEF VEC {O,P,I}(TYPEDEF) macro, to get the non-memory allocation version, and then a DEF VEC ALLOC {O,P,I}(TYPEDEF,ALLOC) macro to get memory managed vectors. Variables of vector type are declared using a VEC(TYPEDEF, ALLOC) macro. The ALLOC argument specifies the allocation strategy, and can be either 'gc' or 'heap' for garbage collected and heap allocated respectively. It can be 'none' to get a vector that must be explicitly allocated (for instance as a trailing array of another structure). The characters O, P and I indicate whether TYPEDEF is a pointer (P), object (O) or integral (I) type. Be careful to pick the correct one, as you'll get an awkward and inefficient API if you use the wrong one. There is a check, which results in a compile-time warning, for the P and I versions, but there is no check for the O versions, as that is not possible in plain C. Due to the way GTY works, you must annotate any structures you wish to insert or reference from a vector with a GTY(()) tag. You need to do this even if you never declare the GC allocated variants. An example of their use would be,

#### **Additional Representations**

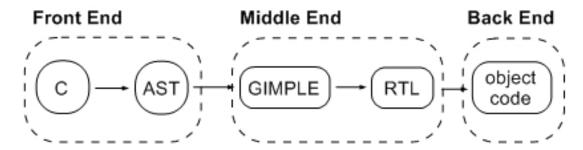


Figure 3 Additional representations of GCC 4.1.

Figure 2<sup>4</sup> shows additional representations of GCC 4.1. Because of the differences in languages, the format of the generated ASTs is slightly different for each language. The next step after AST generation is the unification step in which the AST tree is converted into a unified form called generic. After this, the middle end part of the compiler takes control. First, the tree is converted into another representation called *GIMPLE*. In this form, each expression contains no more than three operands, all control flow constructs are represented as combinations of conditional statements and goto operators, arguments of a function call can only be variables, etc. Figure 2<sup>5</sup> illustrates the differences between a tree in generic form and a tree in GIMPLE form. GIMPLE is a convenient representations for optimizing the source code.

After GIMPLE, the source code is converted into the *static single assignment (SSA)* representation. The central idea of this form is the fact that each variable is assigned to only once, but can be used at the right hand side of an expression many times. Every time the same variable of a tree in the GIMPLE form is reassigned, the compiler creates a new version of that variable and stores the new value into it. When the same variable is assigned to in both branches of a conditional expression, one needs to merge the two possible values of the variable into a single variable. This operation is denoted as PHI function in the SSA form.

The SSA form is also used for optimizations. GCC performs more than 20 different optimizations on SSA trees. After the SSA optimization pass, the tree is converted back to the GIMPLE form.

Take home:

GCC is a compiler collection that consists of a front end for each programming language, a middle end, and a back end for each architecture. The main representations that each source file goes through are AST in the front end, RTL in the middle end, and the assembly representation in the back end. GCC compiles one file at a time.

<sup>4</sup> http://en.wikibooks.org/wiki/File:Gcc\_repr.jpg

 $<sup>\</sup>label{eq:http://en.wikibooks.org/w/index.php?title=File:Gccinternals-img2.png\&action=dit\&redlink=1$ 

#### 3.0.3 GCC Initialization

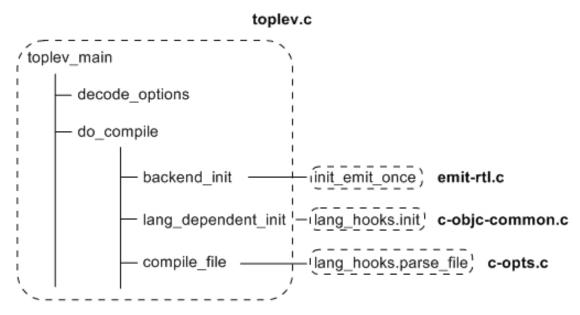


Figure 4 GCC initialization.

The C front end includes the C/C++ preprocessor and the C compiler. Program cc1 includes both the preprocessor and C compiler. It compiles a C source file and generates an assembly (S) file. The compiler frontend and backend interact with each other using callback functions called language hooks. All hooks are included into a global variable struct lang hooks lang hooks that is defined in file langhooks.h. There are the following types of hooks: hooks for tree inlining, hooks for call graph, hooks for functions, hooks for tree dump, hooks for types, hooks for declarations, and language-specific hooks. The default values for the hooks are defined in file langhooks-def.h. GCC initialization consists of command line option parsing, initializing the back end, creating the global scope, and initializing the built-in data types and functions. Each declaration is associated with a scope. For example, a local variable is associated with its function's scope. A global declaration is associated with the global scope. File toplev.c contains the main cc1 function toplev main() and global variables that define the state of the compiler. Variable current function decl is the declaration of the function being compiled or NULL if between functions. Function toplev main() is the function that processes the command-line options, initializes the compiler, compiles a file, and frees up the allocated resources. Function decode options() processes the command-line options and sets the corresponding variables within the compiler. Following the command line option parsing function do compile() is called. It performs the back-end initialization by calling function backend init(). Backend initialization includes a number of steps. Function init emit once() generates RTL expressions for a number of registers: pc rtx for the program counter, cc0 for the condition, stack pointer rtx, frame pointer rtx, etc. They are saved in array global rtl. After that, function  $lang\_dependent\_init()$  performs language-dependent initialization which includes the initialization of a front-end and a back-end. The C initialization function

11

 $c\_objc\_common\_init()$  creates built-in data types, initializes the global scope and performs other initialization tasks. Function  $c\_common\_nodes\_and\_builtins()$  creates predefined types described in file **builtin-types.def**. The standard C types are created at the initialization time. The following table presents several types:

#### GCC builtin types

Variable Name C type
char\_type\_node char
integer\_type\_node int
unsigned\_type\_node unsigned int
void\_type\_node void
ptr\_type\_node void\*

GCC built-in functions are the functions that are evaluated at compile time. For example, if the size argument of a strcpy() function is a constant then GCC replaces the function call with the required number of assignments. The compiler replaces standard library calls with built-in functions and then evaluates them once the function's AST is constructed. In case of strcpy(), the compiler checks the size argument and uses the optimized built-in version of strcpy() if the argument is constant.  $builtin\_constant\_p()$  allows to find out if the value of its argument is known at compile time. GCC builtins are used beyond GCC. For example, the string processing library of Linux kernel uses  $builtin\_constant\_p()$  to invoke the optimized version of a string processing function if the string size is known at compile time. GCC evaluates each builtin function using a corresponding  $expand\_builtin()$  function. For example,  $builtin\_strcmp()$  is evaluated using  $expand\_builtin\_strcmp()$ . The following table gives examples of GCC builtins:

#### GCC builtin functions

Builtin Name Explanation
builtin\_constant\_p returns true if the argument is a constant
builtin\_memcpy equivalent to memcpy()
builtin\_strlen equivalent to strlen()

Take home: GCC initialization consists of command line option parsing, initial-

izing the back end, creating the global scope, and initializing the

built-in data types and functions.

#### 3.0.4 Parser and Preprocessor

Following the initialization, function  $do\_compile()$  calls function  $compile\_file()$ . This function invokes  $parse\_file()$  front-end language hook which is set to function  $c\_common\_parse\_file()$  for C language. The latter function invokes function  $fin-ish\_options()$  which initializes the preprocessor and handles -**D**, -**U**, and -**A** command

line options (which are equivalent to #define, #undef, and #assert respectively). The C preprocessor handles the preprocessor directives such as #define, #include in the source code.

#### Parser

Parser is implemented manually in file **c\_parser.c**. Compared to earlier releases of GCC, the new parser generates a lower level AST. There were special tree codes for loops, for example  $FOR\_STMT$  denoted a for() loop. In this release the loops are represented as conditional statements and gotos, that is trees of codes  $COND\_EXPR$ ,  $LABEL\_EXPR$ , and  $GOTO\_EXPR$ . This probably means that it is impossible to raise AST representation to original source code.

#### Preprocessor

The preprocessor is implemented as a library. The C language lexer function  $c\_lex()$  calls the libcpp function  $cpp\_get\_token()$  which handles the preprocessor keywords. The state of the preprocessor is defined by variable  $cpp\_reader *parse\_in$ . Type  $struct \ cpp\_reader$  contains most importantly the list of text buffers being processed. Each buffer corresponds to a source file (.c or .h). Function  $cpp\_get\_token()$  calls appropriate functions for legitimate preprocessor keywords. For example, when #include is encountered, function  $do\_include\_common()$  is invoked. It allocates a new buffer and places it on top of the buffer stack making it the current buffer. When a new file is compiled the buffer is popped off the stack and the compilation of the old file continues. Function  $do\_define()$  is called whenever a new macro is defined using #define keyword.

**Take home:** The preprocessor takes care of the preprocessor directives, for example #include and #ifdef.

#### 3.0.5 From Source Code to AST

After running the preprocessor GCC constructs an abstract syntax tree (AST) for each function of the source file. An AST is a number of connected nodes of type *struct tree*. Each node has a *tree code* that defines the type of the tree. Macro  $TREE\_CODE()$  is used to refer to the code. Tree codes are defined in files **tree.def** and **c-common.def**. Trees with different tree codes are grouped into *tree classes*. The following tree classes are defined among others in GCC:

#### GCC tree classes

Tree Class	Explanation
'd'	declarations
'<'	comparison
'2'	binary arithmetic

There are two types of trees: *statements* and *expressions*. Statements correspond to the C constructs, for example an expression followed by a ';', a conditional statement, a loop statement, etc. Expressions are what the statements are built up from. Examples of expressions are an assignment expression, an arithmetic expression, a function call, etc. Examples of tree codes are given in the following table:

#### GCC tree codes

Tree Code	Tree Class	Explanation	Operands
$MODIFY\_EXPR$	'e'	an assignment expres-	$\mathrm{TREE}\_\mathrm{OPERAND}(\mathrm{t},\!0)$
		sion	- left hand side;
			$TREE\_OPERAND(t,1)$
			- right hand side;
$CALL\_EXPR$	'e'	function call	$TREE\_OPERAND(t,0)$
			- function
			definition;
			$\mathrm{TREE}\_\mathrm{OPERAND}(\mathrm{t},\!1)$
			- function argu-
			ments;
FUNC-	'd'	variable declaration	$DECL\_SOURCE\_FILE(t)$
$TION\_DECL$			- source file;
			$DECL_NAME(t)$ -
			variable name;
$ARRAY\_TYPE$	't'	array type	$TREE\_TYPE(t)$
			- type of an
			array element;
			$TYPE\_DOMAIN(t)$
			- type of index;
$\mathrm{DECL\_STMT}$	'e'	variable declaration	$\mathrm{TREE}\_\mathrm{OPERAND}(\mathrm{t},\!0)$
			- variable;
			$DECL\_INITIAL(TREE\_OPERAND(t,1$
			- initial value;

In addition to the tree code that defines the type of the tree, a number of operands different for each tree type are available. For example, an assignment expression has two operands which correspond to the left and right sides of the expression. Macro  $TREE\_OPERAND$  is used to refer to the operands. Macro  $IDENTIFIER\_POINTER$  is used to find the name of an identifier that an  $IDENTIFIER\_NODE$  tree represents. The following table presents several tree nodes, their purpose, and their operands. Each tree has a type that corresponds to the type of C expression that it represents For example, the type of the  $MODIFY\_EXPR$  node is the type of the left-hand side operand.  $NOP\_EXPR$  and  $CONVERT\_EXPR$  trees are used for type casting. Tree  $NULL\_TREE$  is equivalent to NULL. Function  $debug\_tree()$  prints out the tree to stderr. When a new identifier is lexed, it is added to the pool of strings that GCC maintains. The tree code of an identifier is  $IDENTIFIER\_NODE$ . When the

same identifier is lexed again, the same tree node is returned. Function  $get\_identifier()$  returns the tree node for an identifier.

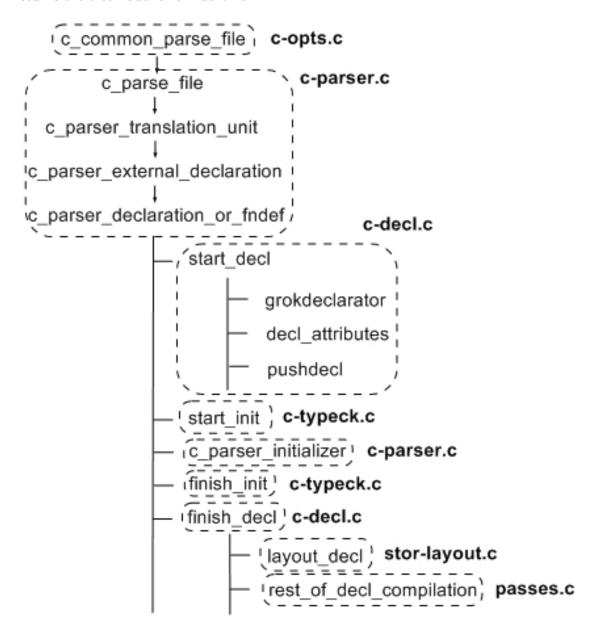


Figure 5 Parsing variable declaration.

A new variable declaration is processed in a number of function calls. First, function  $start\_decl()$  is called with the name of the declaration, its type as returned by the lexer, and its attributes. The function calls grokdeclarator() that checks the type and argument nodes and returns a tree with the code appropriate for the declaration:  $VAR\_DECL$  for variables,  $TYPE\_DECL$  for types, etc. The declaration is then added to the scope. A scope contains all declarations created within a function, but does not contain global declarations. There is also the global scope that contains global declarations. When a function is parsed, its declarations are attached to its body as a BLOCK node. When a declaration is created, the identi-

fier node is associated with the declaration node using IDENTIFIER SYMBOL VALUE. Function lookup name() returns the declaration for a given identifier. When a declaration leaves the scope the tree attributes C DECL INVISIBLE is asserted. A symboltable is not maintained in GCC. Instead, the compiler uses the pool of identifiers and C DECL INVISIBLE attribute. Language hook lang hooks.decls.getdecls() returns the variables in the scope chained together. Functions start init() and finish init() are called for an initialized declaration. Function finish decl() finalizes the declaration. For an array declaration it computes the size of an initialized array. Then function layout decl() is called. It computes the size and alignment of the declaration. Parsing a function depends on the presence of its body. A function declaration is parsed using the same functions as those used for variable declarations. For a function definition, function start function() is called. Then the compiler parses the body of the function. When the function ends function finish function() is called. Functions start decl() and start function() take declaration's attributes parameter as one of their arguments. The attributes are described in the GCC Manual. Attributes are extensions of GNU implementation of C. The following table presents a few of them and explains their purpose:

#### Function attributes

Attribute Explanation

constructor call function automatically before main() destructor call function automatically after main()

alias alias to another function

For each type of C statement there is a function that constructs a tree node of the corresponding type. For example, function <code>build\_function\_call()</code> builds a <code>CALL\_EXPR</code> node for a function call. It takes the identifier of the function name and the arguments as its parameters. The function finds the function declaration using <code>lookup\_name()</code> and type casts the arguments using <code>default\_conversion()</code>. After a function has been parsed, use macro <code>DECL\_SAVED\_TREE</code> to access its body. It is represented with a <code>BIND\_EXPR</code> tree which binds local variables to statements. <code>BIND\_EXPR\_VARS</code> gives a chain of declared variables. <code>BIND\_EXPR\_BODY</code> returns a tree of <code>STATEMENT\_LIST</code> type. The following API allows one to traverse a statement list and to manipulate it:

#### Tree construction API

Function	Purpose
$tsi\_start(stmt\_list)$	get an iterator pointing at list head
$tsi\_last(stmt\_list)$	get an iterator pointing at list tail
$tsi\_end\_p(iter)$	is end of list?
$tsi\_stmt(iter)$	get current element
tsi_split_statement_list_before(&iter)	split elements at iter
tsi_link_after(&iter, stmt, mode)	link chain after iter
$tsi\_next(\&iter)$	next element of the list
append_to_statement_list(tree, &stmt_list)	append tree to stmt_list

It is possible to instrument function prolog/epilog at this level as demonstrated in  $gim-plify\_function\_tree()$ . To add a statement to function epilog, use  $TRY\_FINALLY\_EXPR$ 

tree. Its first operand are the old statements, the second argument is the epilog statements. This type of treet instructs the following passes to execute the statements when a common exit basic block of a function is created. To instrument function prolog, prepend the tree with the desired statements. Therefore,  $BIND\_EXPR\_BODY$  will have the prolog and the  $TRY\_FINALLY\_EXPR$  tree. The ASTs are then converted into the SSA and eventually to the RTL representations. Whether the conversion occurs after parsing each function or when the whole file is parsed is controlled by the compiler option **-funit-at-a-time**. It is false by default.

Take home:

GCC parser builds the AST representation of the source file. ASTs are built up of tree nodes. Each node has a code. Tree nodes correspond to the statements and expressions of C. Function debug tree() prints out the tree.

#### 3.0.6 From AST to GIMPLE

An AST is gimplified when gimplify\_function\_tree() is eventually called from finish\_function(). GIMPLE representation is based on SIMPLE described in [2]<sup>6</sup> According to this paper, the goal is to represent the tree as basic statements:

#### **GIMPLE** trees

x=a binop b	x=a	x=cast b	f(args)
*p=a binop b	*p=a	*p=cast b	-
x=unop a	x=*q	x=&y	x=f(args)
*p=unop a	p=*q	*p=&y	p=f(args)

Temp variables are created as necessary in functions create\_tmp\_var() and declare\_tmp\_vars(). At this stage, GCC optimizes of complex conditional expressions, that is

```
if (a || b) stmt;
is translated into

if (a) goto L1;
if (b) goto L1; else goto L2;
L1:
stmt;
```

Also, each branch of a conditional expression is wrapped into STATEMENT\_LIST tree.

#### 3.0.7 From GIMPLE to RTL

Register Transfer Language represents an abstract machine with an infinite number of registers. Type rtx describes an instruction. Each RTL expression has a code and machine

mode. Similarly to ASTs, codes are grouped in a number of classes. They are defined in mode-classes.def.

#### Classes of RTL expressions

Classes	Explanation
$RTX\_CONST\_OBJ$	represent a constant object (e.g, CONST_INT)
$RTX_OBJ$	represent an object (e.g, REG, MEM)
RTX_COMPARE	comparison (e.g, LT, GT)
RTX_COMM_COMPARE	commutative comparison (e.g, EQ, NE, ORDERED)
$RTX_UNARY$	unary arithmetic expression (e.g, NEG, NOT)
$RTX\_COMM\_ARITH$	commutative binary operation (e.g., PLUS, MULT)
$RTX\_TERNARY$	non-bitfield three input operation (IF_THEN_ELSE)
RTX_BIN_ARITH	non-commutative binary operation (e.g., MINUS, DIV)
RTX_BITFIELD_OPS	bit-field operation (ZERO_EXTRACT,
	SIGN_EXTRACT)
RTX_INSN	machine insn (INSN, JUMP_INSN, CALL_INSN)
$RTX\_MATCH$	something that matches in insns (e.g, MATCH_DUP)
RTX_AUTOINC	autoincrement addressing modes (e.g. POST_DEC)
$RTX\_EXTRA$	everything else

Machine modes listed in file machmode.def specify a size and format of data at the machine level. At the syntax tree level, each ...\_TYPE and each ...\_DECL node has a machine mode which describes data of that type or the data of the variable declared. A list of instructions is built when function is compiled. Function emit\_insn() adds an instruction to the list. A variable declaration AST has its RTL already generated. Use DECL\_RTL to access it. Function emit\_cmp\_and\_jump\_insns() outputs a conditional statement. emit\_label() prints out a label. These functions chain instructions one after another. Macros PREV\_INSN and NEXT\_INSN are used to traverse the list. It is possible to access the first and last instructions using first\_insn and last\_insn. get\_insns() gives the first insn of the current sequence or current function. Use debug\_rtx() to print out an RTL instruction on the screen and function print\_rtl() to print a list of RTL expressions. A number of functions create the nodes. For example, gen\_label\_rtx() build a label. The most generic functions are located in the target-specific directory. For example, x86 architecture rtl generation files genrtl.c and genrtl.h are in ./host-i686-pc-linux-gnu.

#### 3.0.8 From RTL to Object

Each target architecture has its description represented as struct gcc\_target targetm. Default initializers are available in file targhooks.c The backend generates the assembly code for the specified target platform. Function  $output\_asm\_insn()$  is called for each instruction written to the assembly file. Function  $final\_start\_function()$  generates the prolog of the function before it is saved to the assembly file.

#### 3.0.9 Lowering Passes

The processing of a function includes its *lowering* when a number of optimization passes are applied to it in function tree\_lowering\_passes(). As a result of lowering a function its control-flow graph is generated. A subsequent call to function cgraph\_create\_edges() uses basic block information to augment edges of the callgraph with invocations that the current function performs. References to functions yet undefined are saved in function record references().

#### All lowering passes

Name Meaning N/Aremove useless stmts mudflap 1 narrow-pointer bounds-checking by tree rewriting lower cf Lowers GIMPLE into unstructured form pass\_lower\_eh Exception handling semantics and decomposition for trees pass build cfg Create basic blocks pass lower complex O0 complex operation lowering without optimization pass\_lower vector Lower vector operations to scalar operations pass warn function return Emit return warnings pass\_early\_tree\_profile set up hooks for tree-based profiling

#### 3.0.10 Switch Statement

Let us consider how a switch statement is translated from the source code to GIMPLE to RTL. Function c parser switch statement() is called when the statement is encountered in the source code. A typical switch statement has a number of case statements that might have breaks. Therefore, the parser has c break label tree that marks the place where switch ends. The function parses the body of the statement and generates LABEL EXPR tree for the break label if at least one break was found. Function c finish case() attaches the body to the SWITCH EXPR tree as one of its operands. In addition, this tree has two other operands: switch condition and switch labels. The operands are accessed using macro SWITCH COND(), SWITCH BODY(), and SWITCH LABELS(). labels are not filled in at the parse time. A switch statement is gimplified in function gimplify switch expr(). The idea is to separate the body from the decision part and generate switch labels to make it possible to redirect execution to the appropriate case after verifying the condition. We will consider the case when default label exists. This function has two pointers to the statement list: pre\_p which is the list of side effects and expr\_p which is the statement itself. The body of the switch is gimplified in gimplify to stmt list(). Case labels are saved in field case labels of variable struct gimplify ctx gimplify ctxp. Then the function creates a TREE\_VEC of labels and initializes them with the corresponding case label. The TREE VEC is assigned to SWITCH LABELS operand of switch statement which is then appended to the pre p list. The original statement is then overwritten with the SWITCH BODY using expr p pointer. Finally, the SWITCH BODY operand of the switch statment in the side effects list is wiped out so that it contains labels only. From this point on it is clear that compiler tries to represent the original statement using a jump table which maps each possible index value to the address of the corresponding case. Function expand\_case() implements this idea. It generates a table\_label at which jump instructions for each possible index value are generated. Then function try\_tablejump() is called which expands index tree into index rtl and invokes do\_tablejump(). This function generates absolute index rtl which combines the base address table\_label and index offset. It emits jump instruction to the proper entry of the jump table afterwards. The execution continues in function expand\_case(). The jump table is generated using SWITCH\_LABELS:

```
labelvec[i] = gen_rtx_LABEL_REF (Pmode, label_rtx (n->code_label));
```

A number of jump instructions are emitted finally:

**Take home:** The backend generates the assembly code for the specified target platform.

- 1. ^7 8
- 2. ^9 L. Hendren, C. Donawa, M. Emami, G. Gao, Justiani, and B. Sridharan. *Designing the McCAT Compiler Based on a Family of Structured Intermediate Representations.* In Proc. of the 5th International Workshop on Languages and compilers for Parallel Computing, 1992.

<sup>7 #</sup>ref\_rh\_magazine

<sup>8</sup> http://www.redhat.com/magazine/002dec04/features/gcc/

<sup>9 #</sup>ref\_simple\_paper

# 4 Compilation of a function

# 4.1 Function Prolog and Epilog

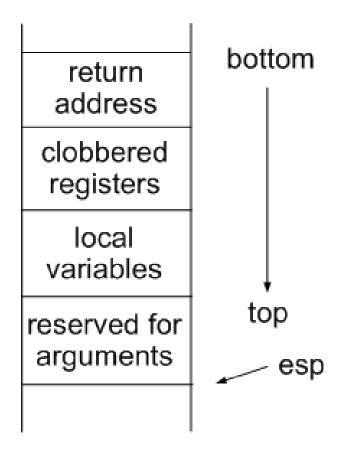


Figure 6 GCC stack frame layout.

A prolog is used to initialize the function, for example set up its stack frame. A function epilog is used to finalize execution. At each level of compilation, there are common actions performed for each function. Therefore, a number of prolog/epilog generation functions exist in GCC. At the GIMPLE level prolog is generated in function gimplify\_function\_tree(). It adds profiling hooks if instructed so using -finstrument-functions. At the RTL level, function expand\_function\_start is used. It starts the RTL for a new function, and set variables used for emitting RTL. It also makes the label for return statements to jump to and decides whether to return the value in memory or in a register. After this, it calls assign\_parms() which maps the formal-ins to the actual-ins. Machine-level prolog and epilog are added in pass flow2, function thread\_prologue\_and\_epilogue\_insns() which relies upon machine

description file, for example i386.md. The description has a number of entries. The prolog and epilog entries is used in this case. They are set to ix86\_expand\_prologue() and ix86\_expand\_epilogue() respectively that generate RTL prologue and epilog for i386 architecture. The machine-specific code takes care of registered used in the function so that they are preserved between function calls. The push instructions are added as the very first instructions of a function. The machine-specific code is generated as an RTL expression rather than assembly because this is the representation used at this pass of the compiler. The RTL representation is the most general across various targets. Therefore, a push instruction of i386 corresponds to a number of RTL expressions:

At the assembly output level, prolog is generated in functions assemble\_start\_function() and final\_start\_function(). Their output is mostly related to debugging. One should note that the prolog functions are executed from the higher-level representation to the lower-level ones. At the run-time, the execution order of the added code is opposite. That is, the machine-level prolog is executed first. In case of i386, it saves the registers used in the function and sets up a new stack frame. Then the parameters are received and the profiling hooks are called.

# 4.2 Local Control Flow Analysis

A control-flow graph is built up of basic blocks which is a sequence of instructions with only one entry and one exit. Type struct basic\_block describes it. There is a pointer to the statement list that defines the basic block. Each basic block also has a pointer to the previous and next BB. Therefore, it is possible to link them in a list. Fields preds and succs give access to control/data flow edges into and out of the block. Function create\_basic\_block() takes the first and the last statements and inserts the new BB after a given one. make\_edge() links the two BBs. Use macro FOR\_EACH\_BB to traverse a CFG. Each function flow graph has an entry and an exit block accessed using ENTRY\_BLOCK\_PTR and EXIT\_BLOCK\_PTR respectively. Function build\_tree\_cfg() takes a GIMPLE tree and generates the CFG. First, it initializes the CFG with two BBs: ENTRY\_BLOCK and EXIT\_BLOCK. It calls make blocks(), then make edges().

# 5 Function calls

### 5.1 Global Control-Flow Analysis

The functions of a file are used to generate a *callgraph* in file cgraphunit.c. The two relevant functions are cgraph\_finalize\_compilation\_unit() which is called from function pop\_file\_scope() after the file has been parsed and cgraph\_finalize\_function() which is called from finish\_function(). The effect of each function depends on the compilation mode. The **unit-at-a-time** mode instructs the compiler to build the callgraph only after each function has been parsed. When this option is not present a function is converted as soon as it is parsed. cgraph\_finalize\_function() calls cgraph\_analyze\_function() that converts it to RTL. Otherwise, the function is queued in cgraph\_nodes\_queue. At the end cgraph\_finalize\_compilation\_unit() takes care of the queue. cgraph\_nodes is the global variable representing the callgraph. Function dump\_cgraph allows one to print out the callgraph.

# 5.2 Parameter Passing

In this chapter we will find out how functions call each other. Typically, a function passes a number of parameters when it makes a call. A stack frame is created when the function begins. It is possible, however, that the stack frame of the previous function gets reused. This type of function call is called a sibling call. When the function body is not large enough the run-time overhead of setting up a stack frame is too high. In this case the callee function gets inlined into the parent function. Function expand\_call() takes a CALL\_EXPR tree and generates RTL expression. It has to decide argument passing mode. struct arg\_data contains necessary information for each argument.

#### struct arg data

Field Name Explanation

tree tree\_value Tree node for this argument

enum machine mode mode Mode for value

rtx value Current RTL value for argument, or 0 if it isn't

precomputed

rtx initial\_value Initially-compute RTL value for argument; only

for const functions.

rtx reg Register to pass this argument in, 0 if passed

on stack

rtx tail call reg Register to pass this argument in when gener-

ating tail call sequence

struct arg data	
rtx parallel value	If REG is a PARALLEL, this is a copy of
	VALUE pulled into the correct form for
	emit_group_move.
int unsignedp	If REG was promoted from the actual mode of
	the argument expression, indicates whether the
	promotion is sign- or zero-extended
int partial	Number of bytes to put in registers. 0 means
	put the whole are in registers or not passed in
	registers.
int pass_on_stack	Nonzero if argument must be passed on stack.
	Note that some arguments may be passed on
	the stack even though pass_on_stack is zero,
	just because FUNCTION_ARG says so
struct locate_and_pad_arg_data	Some fields packaged up for lo-
locate	cate_and_pad_parm
rtx stack	Location on the stack at which parameter
	should be stored
rtx stack_slot	Location on the stack of the start of this argu-
	ment slot
rtx save_area	Place that this stack area has been saved, if
	needed
rtx *aligned_regs	If an argument's alignment does not permit
	direct copying into registers, copy in smaller-
	sized pieces into pseudos. These are stored in a

Generating a function call is impossible without certain machine-specific information, for example the number of hardware registers of different types. A number of macros defined in each architecture .h file take care of connecting the middle-end with the back-end:

block pointed to by this field.

says how many word-sized pseudos we made

#### **Argument macros**

int n aligned regs

Macro Name	Explanation
INIT_CUMULATIVE_ARGS	Initialize CUMULATIVE_ARGS data structure for
	a call to a function whose data type is FNTYPE.
FUNCTION_ARG	Define where to put the arguments to a function.
	Value is zero to push the argument on the stack, or
	a hard register in which to store the argument.
FUNC-	Update the data in CUM to advance over an argu-
TION ARG ADVANCE	ment.

Function init\_cumulative\_args() in file i386.c takes care of this in case of x86 architecture. It takes into account function attributes regparm and fastcall that a user might specify in which case the number of available registers is set accordingly. However, if the function takes a variable number of arguments then all parameters are passed in the stack. Parameters

ters' location is decided in function initialize\_argument\_information(). A machine-specific function arg() returns the rtl for an argument if it goes into a register:

```
ret = gen_rtx_REG (mode, regno);
```

It is possible that a parameter is passed both on the stack and in a register, for example if the parameter's type is addressable. Depending on certain conditions, sibling call instruction chain is generated in addition to the normal chain. Let us consider the case when only normal chain is generated. Variable rtx argblock is the address of space preallocated for stack parameters (on machines that lack push insns), or 0 if space not preallocated. A number of machine-specific variables shape up the stack. ACCUMULATE\_OUTOING\_ARGS instructs the compiler to preallocate the sufficient number of bytes for all arguments of any function call in the function prolog. After that, function arguments are saved in that area without modifying stack frame size. ACCUMULATE\_OUTOING\_ARGS depends on variable target\_flags. It depends on machine configuration and command-line options. In case of ACCUMULATE\_OUTOING\_ARGS, i386-specific variable

```
const int x86_accumulate_outgoing_args = m_ATHLON_K8 | m_PENT4 |
m_NOCONA | m_PPRO;
```

and a command-line option **-maccumulate-outgoing-args** enable this feature. This means that it is enabled on a Pentium4 and that push/pop instructions are not used to pass funcitons' parameters. If we preallocated stack space, compute the address of each argument and store it into the ARGS array. Precompute parameters as needed for a function call. this routine fills in the INITIAL\_VALUE and VALUE fields for each precomputed argument. precompute\_arguments() Given a FNDECL and EXP, return an rtx suitable for use as a target address in a call instruction

```
funexp = rtx_for_function_call (fndecl, addr);
```

Precompute all register parameters. It isn't safe to compute anything once we have started filling any specific hard regs. precompute\_register\_parameters (num\_actuals, args, &reg\_parm\_seen); Now store (and compute if necessary) all non-register parms. These come before register parms, since they can require block-moves, which could clobber the registers used for register parms. Parms which have partial registers are not stored here, but we do preallocate space here if they want that.

Do the register loads required for any wholly-register parms or any parms which are passed both on the stack and in a register. Their expressions were already evaluated.

Finally, emit\_call\_1() generates instructions to call function FUNEXP, and optionally pop the results. The CALL\_INSN is the first insn generated. When the placement of the arguments is decided variable struct args\_size args\_size saves the total size of stack arguments. It records the size of a sequence of arguments as the sum of a tree-expression and a constant. The tree part is necessary to handle arguments of variable size, for example arrays arguments which size is not known at compile-time. The C language does not allow variable-sized arguments. One might wonder how the callee function finds out where the arguments arrive. It also uses machine-specific information that the caller used. Rerunning INIT\_CUMULATIVE\_ARGS, FUNCTION\_ARG, and FUNCTION\_ARG\_ADVANCE in the callee decides whether an argument should arrive in a register or on the stack identically to expand\_call.

# 6 Stackguard

: \_\_strcpy\_ichk (dest, src))

Stackguard is implemented in the following files. These files translates GIMPLE to RTL. They take advantage of the CFG to instrument function prolog and epilog. 1) cfgexpand.c Stackguard is created in function expand used vars(). Depending on stackguard flag either all arrays or character arrays only are allocated first. tree expand cfg() takes care of prolog when it calls stack protect prologue(). The corresponding function stack protect epilogue() is called from calls.c to take care of tail calls and from tree expand cfg(); 2) function.c stack protect prologue(), stack protect epilogue() The functions use machine definition targetm. 3) targhooks.c Contains default initializers of target architecture hooks. default stack protect guard() builds a VAR DECL tree node that represents variable stack chk guard. default stack protector fail() builds a call to function stack chk fail(). These ASTs are converted into RTL trees using expand expr stmt(). The prolog and epilog functions add RTL expressions directly as well. For example, to detect a compromise function epilog generates a conditional statement that compares the canary word on the stack with its initial value:

```
emit_cmp_and_jump_insns (x, y, EQ, NULL_RTX, ptr_mode, 1, label);
where x and y are corresponding declaration RTLs:

x = validize_mem (DECL_RTL (cfun->stack_protect_guard));
y = validize_mem (DECL_RTL (guard_decl));
```

Stackguard implements a certain amount of static analysis to make sure that suspicious functions with arguments of fixed length do not have vulnerabilities. The underlying mechanism is that of builtins. Stackguard has file strcpy.h that defines functions of interest, for example strcpy(). The new definition uses builtins \_\_ssp\_bos() to find out object size and \_\_builtin\_\_\_strcpy\_chk():

The builtins are evaluated at compile time. Function warning() called in maybe emit chk warning() prints out a message if the destination buffer is overflown.

In case when Stackguard is unable to find out statically if the buffer is overrun function <code>\_\_strcpy\_chk()</code> is called at run-time. Library libssp provides the function. However, the protected program does not require linking with this library explicitly. Instead, GCC instructs the loader to use a shared library.

# 7 GEM Framework

#### **7.0.1** Hooks

GEM framework is designed to facilitate development of compiler extensions. The idea of GEM is similar to the idea of Linux Security Modules¹ (LSM), a project that defines hooks throughout Linux kernel that allow one to enforce a security policy. GEM defines a number of hooks throughout GCC's source code. It is implemented as a patch to GCC. With GEM, a compiler extension is developed as a stand-alone program. It is compiled into a dynamically-linked module which is specified as the command line argument when GCC is invoked. GCC loads the module and calls its initialization function. The module then registers its hooks that are call-back functions in GCC. In addition to the compiler hooks, GEM provides macros and functions that simplify extension development. In this chapter we will first introduce the hooks that GEM framework adds to GCC. Then we describe the typical issues in extension programming. The project home page is at <sup>2</sup> GEM adds several hooks throughout GCC source code. New hooks are added to GEM as necessary.

- Hook gem\_handle\_option to function <code>handle\_option()</code> which processes each command line option. The hook takes the current option as its argument. If the hook returns value GEM RETURN then GCC ignores the option.
- Hook **gem\_c\_common\_nodes\_and\_builtins** which is called after all standard types are created. The GCC extension can create additional types.
- Hook **gem\_macro\_name** allows one to save the name of the macro being defined. Another GEM hook **gem\_macro\_def** is called when the macro definition is parsed. Using the macro name of the new macro definition it is possible to re-define the macro. This hook is added to function *create iso definition()*.
- Hooks **gem\_start\_decl** and **gem\_start\_function** are called when a function or variable declaration/definition starts.
- Hook **gem\_build\_function\_call** allows one to modify the name and the arguments of a function call.
- Hook **gem\_finish\_function** is inserted to *finish\_function()* which is called from from grammar file. The compiler extension receives the function body of the function before it is translated into RTL.
- Hooks **gem\_output\_asm\_insn** and **gem\_final\_start\_function** are added to function  $output\_asm\_insn()$  which is called for each instruction of the assembly code and

<sup>1</sup> http://lsm.immunix.org/

<sup>2</sup> http://research.alexeysmirnov.name/gem

function final\_start\_function() called when the assembly code is written to the file, respectively. The former hook receives the text that is written to the file which allows it to modify the output. The latter hook can modify function's prolog.

**Take home:** GEM hooks are defined mostly at the AST level. A few hooks are defined at the assembly level. The new hooks are added as necessary.

#### 7.0.2 Traversing an AST

When the function's AST is constructed one can instrument it. GEM's gem\_finish\_function hook receives the AST of a function. The idea is to traverse the AST and instrument the AST nodes as necessary. Function  $walk\_tree()$  takes the AST, the callback function, the optional data, NULL by default, and the walk\_subtrees parameter, NULL by default. The callback function is called for each node of the AST before the operands are traversed. If the callback function modifies the  $walk\_subtree()$  variable then the operands are not processed. The following code demonstrates the idea:

```
static tree walk_tree_callback(tree *tp, int *walk_subtrees, void *data) {
   tree t=*tp;
   enum tree_code code = TREE_CODE(t);
   switch (code) {
   case CALL_EXPR:
     instrument_call_expr(t);
     break;
   case MODIFY_EXPR:
     instrument_modify_expr(t);
     break;
   }
}
```

**Take home**: Function walk\_tree() traverses an AST applying user-defined callback function to each tree node.

#### 7.0.3 Instrumenting an AST

In this section we describe functions that create new tree nodes and how to add the new nodes to an AST.

#### Lookup of a Declaration in the Symbol Table

```
void gem_find_symtab(tree *t_var, char *name) {
  tree t_ident = get_identifier(name);
  if (t_ident) *t_var = lookup_name(t_ident); else *t_var=NULL_TREE;
```

#### **Building Tree Nodes**

The walk\_tree callback function can instrument the AST. Functions build1() and build() construct new tree nodes. The former function takes one operand, the latter one takes more then one operand. The following code computes the address of the operand, same as '&' C operator:

```
t = build1(ADDR_EXPR, TREE_TYPE(t), t);
```

The following example refers to an array element arr[0]:

```
t = build(ARRAY_REF, integer_type_node, arr, integer_zero_node);
```

The following example builds an integer constant:

```
t = build_int_cst(NULL_TREE, 123);
```

Building a string constant is more difficult. The following example demonstrates the idea:

```
tree gem_build_string_literal(int len, const char *str) {
   tree t, elem, index, type;
   t = build_string (len, str);
   elem = build_type_variant (char_type_node, 1, 0);
   index = build_index_type (build_int_2(len-1, 0));
   type = build_array_type (elem, index);
   T_T(t) = type;
   TREE_READONLY(t)=1;
   TREE_READONLY(t)=1;
   TREE_CONSTANT(t)=1;
   type=build_pointer_type (type);
   t = build1 (ADDR_EXPR, type, t);
   t = build1 (NOP_EXPR, build_pointer_type(char_type_node), t);
   return t;
}
```

To build a function call one needs to find the function's declaration and build the list of arguments. Then the CALL EXPR is constructed:

```
gem_find_symtab(&t_func_decl, "func");
t_arg1 = build_tree_list(NULL_TREE, arg1);
t_arg2 = build_tree_list(NULL_TREE, arg2);
...
TREE_CHAIN(t_arg1)=t_arg2;
...
TREE_CHAIN(t_argn)=NULL_TREE;
t_call = build_function_call(t_func_decl, t_arg1);
```

If you want to build a list of statements  $\{ \text{ stmt1}; \text{ stmt2}; \dots \}$ , then you need to use function append to statement list():

```
tree list=NULL_TREE;
for (i=0; i<num_stmt; i++) {
  BUILD_FUNC_CALL1(t_call, t_send, t_arr[i], NULL_TREE);
  append_to_statement_list(t_call, &list);</pre>
```

}

#### Adding Nodes to a Tree

GCC 4.1 has an interface that allows one to add a chain of nodes into another chain of nodes implemented in file **tree-iterator.c**. Functions  $tsi\_start()$  and  $tsi\_last()$  create a tree statement iterator and assigns it to the first or the last tree in the list, respectively. Functions  $tsi\_link\_before()$  and  $tsi\_link\_after()$  link a statement using the iterator either before or after the current statement. There is also function  $append\_to\_statement\_list()$  that adds a node to a list. If the specified list argument is  $NULL\_TREE$  then a new statement list is allocated.

#### **Building Function and Variable Declarations**

A global declaration is added in hook gem\_c\_common\_nodes\_and\_builtins(). In this following example we build a structure type and create a global variable of this type. The structure has a field of type unsigned int and a function pointer field.

```
t_log = make_node(RECORD_TYPE);
decl_chain = NULL_TREE;
field_decl = build_decl(FIELD_DECL, get_identifier("addr"),
unsigned_type_node);
TREE_CHAIN(field_decl)=decl_chain;
decl_chain=field_decl;
DECL_FIELD_CONTEXT(decl_chain) = t_log;
t_func_type = build_function_type_list(void_type_node, unsigned_type_node,
NULL_TREE);
field_decl = build_decl(FIELD_DECL, get_identifier("add_addr"),
build_pointer_type(t_func_type);
TREE_CHAIN(field_decl)=decl_chain;
decl_chain=field_decl;
DECL_FIELD_CONTEXT(decl_chain) = t_log;
TYPE_FIELDS(t_log) = nreverse(decl_chain);
layout_type(t_log);
pushdecl(build_decl(TYPE_DECL, get_identifier("log_t"), t_log));
 decl = build_decl(VAR_DECL, get_identifier("log"), build_pointer_type(t_log));
DECL_EXTERNAL(decl)=1;
pushdecl(decl);
```

#### 7.0.4 When to Instrument

In this section we will describe when each of GEM hooks is used.

- Add new function and type declarations in hook gem\_c\_common\_nodes\_and\_builtins.
- Instrument an AST after it is parsed in hook **gem** finish function.
- Modify attributes of a declaration in hooks **gem\_start\_decl** and **gem\_finish\_decl**. Let us say we would like to replace local array declarations char arr[10] with a heap array char \*arr=(char\*)malloc(10);

```
void 12h_start_decl(void *p_decl, void *p_declspecs, init initialized, void
*p_attr) {
   struct c_declarator *decl = *((struct c_declarator**)p_decl);
   if (current_function_decl == NULL_TREE) return;
   if (decl->kind == cdk_array) {
      decl->kind = cdk_pointer;
      decl->u.pointer_quals = 0;
   }
}

void 12h_finish_decl(tree decl, tree *init, tree spec) {
   ...
   gem_find_symtab(&t_malloc, "malloc");
   BUILD_FUNC_CALL1(t_call, t_malloc, build_int_cst(NULL_TREE, size),
   NULL_TREE);
   *init = build1(NOP_EXPR, build_pointer_type(char_type_node), t_call);
   DECL(decl) = build_int_cst(NULL_TREE, 0); // if this field is NULL the init is ignored
}
```

• Replace function call with a proxy function

#### 7.0.5 Function Prolog/Epilog

The assembly instructions are written to the assembly file:

**Take home:** Assembly instructions are added to function prolog and epilog using hooks gem output asm insn and gem final start function.

## 8 C Function Overloading

#### 8.1 Function overloading in C

The C function overloading extension aims at including a C++ feature to C language which allows use functions with the same name but different argument types. The cfo/test.c example of GEM demonstrates this feature:

```
void ec_aa_add(int from, char *to);
void ec_aa_add(int from, int to);
```

are used to add an element to a pool data structure. The idea behind the implementation of the extension is to rewrite each function declaration so that the new name includes the type information of function's arguments. In the above case, the modified names are ec\_aa\_add\_int\_char\_ptr and ec\_aa\_add\_int\_int. The compilation continues normally with the updated names. Because of the above modification, the call names need modification as well. The renaming takes into account the types of the arguments so that the appropriate function is called. For example, the compiler will modify

```
ec_aa_add(1,2);

to
ec_aa_add_int_int(1,2);
```

Three hooks are used in this extension. Functions cfo\_start\_decl() and cfo\_start\_function() intercept declarations. They call cfo\_alias\_decl() that replaces the name using argument types. To preserve the library code for the programs that do not use CFO extension, the following technique is used. If the function name is encountered first time, an alias is created to its old name so that one can invoke the function using either of them. Thus, the legacy code will use the old name, while the CFO-compiled code will use the type-augmented name. Finally, the declaration name is updated to include type information. Hook cfo\_build\_function\_call() is invoked from the parser when a new function call is found. It replaces the name of callee function with the name that takes into account the types of actual-in arguments.

## 9 Return Address Defense

#### 9.1 Return Address Defense

A prerequisite to understanding the proposed extension is the following paper: Tzi-Cker Chiueh, Fu-Hau Hsu, "RAD: A Compile-time Solution to Buffer Overflow Attacks," 21st IEEE International Conference on Distributed Computing Systems (ICDCS), Phoenix, Arizona, USA, April 2001. <sup>1</sup> The paper proposed to instrument function prolog and epilog. The return address is saved in a well-protected buffer so that overwriting stack frame does not hurt it. The copy of the address is cross-checked with the one on the stack to detect a possible attack. Implementing the proposed method has a number of difficulties. Different architectures have different stack frame layout. For example, the x86 prolog has the code to save clobbered registers which is generated in the machine-specific component of GCC. Therefore, instrumenting prolog at the RTL level will not take into account those registers because their number is unknown at that time. There are two approaches to this problem. The first is to instrument prolog at the machine level which is not portable across different architectures. The second approach is to instrument the call-site instead. The idea of the proposed implementation is to find the approximation of the return address using the address of a code label immediately following the call instruction. It is also possible to find the address on the stack where the return address is written using the number of actual arguments that are pushed on the stack. In case of x86 frame layout, the space is reserved at the frame creation time, therefore the stack pointer not modified when a function is called. The return address approximation and the stack pointer are saved in a well-protected buffer, similarly to the original RAD approach. The following code snippet illustrates the idea:

```
func1() {
    ...
    jmp L2
L1: // the return address is on stack
    push $esp // we also need stack pointer
    rad_prolog() // save the two arguments in a safe buffer
    add $esp, 8 // make sure no traces left
    func2(); // the original call-site
    jmp L3
L2: call L1
L3: ....
```

When the instrumented code is executed, the control flow is directed to a label after the original call instruction. At that point, a call instruction saves the current EIP value and directs the control flow back to label L1. Therefore, the return address approximation argument is already on stack. We also need to push the ESP as the second argument. Function rad\_prolog adds the pair to a safe buffer. The arguments are popped and the

<sup>1</sup> http://www.ecsl.cs.sunysb.edu/tr/TR96.pdf

original function is called. The added call instruction is jumped over upon return. We also need to instrument function epilog. GIMPLE level works in this case because there is a TRY\_FINALLY\_EXPR code. The most recent pair of return address and its location in the well-protected buffer are used to validate the return address on the stack. An alarm is generated in case the two values mismatch. This extension does not use any machine-specific information. Therefore, it is portable across different architectures, even though we were unable to test it under anything except x86.

## 10 Adding Syntactic Sugar

## 10.1 toString() method for each structure as in Java

#### 10.2 Invoke a block of code from a function as in Ruby

Linux implementation of lists allows to invoke a block of code on each element of the list: pagelist.c<sup>1</sup>:

list\_for\_each\_prev takes the code in the brackets as a parameter. The trick is to use a macro that rolls out to a for() loop whose body becomes the code in the brackets. The goal of this project is to allow programmers to use code blocks in function calls.

# 10.3 Dereference function results when a structure is returned

C allows one to dereference the return value of a function if it is a pointer to a structure:

```
get_struct()->field=0;
```

If the function returns the structure, not a pointer to it, then a compile-time error is generated:

```
get_struct().field=0;
> request for member `field' in something not a structure or union
```

This extension addresses the problem of dereferencing structures that are return values.

<sup>1</sup> http://lxr.linux.no/source/fs/nfs/pagelist.c?v=2.4.20#L199

#### 10.4 Use functions to initialize a variable

When a variable is defined and initialized the initializer is constant. You will get an error if you try to use a function, no matter what this function is:

```
int getint() { return 1; }
int i=getint();
> initializer element is not constant
```

When a variable is used the function it was initialized with is called.

### 10.5 Default values of function arguments as in C++

```
void func(int a=0) {
  printf("a=%d\n", a);
}
int main() {
  func();
}
> syntax error before '=' token
```

## 10.6 Reference parameters as in C++

```
void test(int &a, int &b);
int x,y;
test(x,y);
```

## 10.7 GCC switches in object file

GCC switches in object file<sup>2</sup>

## 10.8 Type information at run-time

There is no type information in C language available at run-time. The idea is to allow program to get the names and offsets of a structure's field at run time, the symbolic name of an enum declaration, etc. For example, instead of writing

```
enum tree_code code;
```

 $<sup>2 \</sup>qquad \texttt{http://gcc.gnu.org/wiki/Record\%20GCC\%20command\%20line\%20switches\%20in\%20object\%20files}$ 

```
switch (code) {
  case VAR_DECL:
    printf("VAR_DECL\n");
    break;
  case BLOCK:
    printf("BLOCK\n");
    break;
    ...
}

one could write

printf("%s\n", type_info(code).name);
```

# 11 Improving Code Style

These exercises are from <sup>1</sup>

#### 11.1 Break up enormous source files

Not terribly hard. Watch out for file-scope globals. Suggested targets:

#### Big Files

File Size	File Name
494K	java/parse.y
413K	combine.c
408K	dwarf2out.c
375K	$\mathrm{cp}/\mathrm{pt.c}$
367K	fold-const.c
356K	loop.c

There are several other files in this size range, which I have left out because touching them at all is unwise (reload, the Fortran front end). You can try, but I am not responsible for any damage to your sanity which may result.

## 11.2 Break up enormous functions

This is in the same vein as the above, but significantly harder, because you must take care not to change any semantics. The general idea is to extract independent chunks of code to their own functions. Any inner block that has a half dozen local variable declarations at its head is a good candidate. However, watch out for places where those local variables communicate information between iterations of the outer loop! With even greater caution, you may be able to find places where entire blocks of code are duplicated between large functions (probably with slight differences) and factor them out.

<sup>1</sup> http://gcc.gnu.org/projects/beginner.html

### 11.3 Break up enormous conditionals

Harder still, because it's unlikely that you can tell what the conditional tests, and even less likely that you can tell if that's what it's supposed to test. It is definitely worth the effort if you can hack it, though. An example of the sort of thing we want changed:

```
if (mode1 == VOIDmode
    || GET_CODE (op0) == REG || GET_CODE (op0) == SUBREG
    || (modifier != EXPAND_CONST_ADDRESS
        && modifier != EXPAND_INITIALIZER
        && ((mode1 != BLKmode && ! direct_load[(int) mode1]
             && GET_MODE_CLASS (mode) != MODE_COMPLEX_INT
             && GET_MODE_CLASS (mode) != MODE_COMPLEX_FLOAT)
            /* If the field isn't aligned enough to fetch as a memref,
               fetch it as a bit field.
            || (mode1 != BLKmode
                && SLOW_UNALIGNED_ACCESS (mode1, alignment)
                && ((TYPE_ALIGN (TREE_TYPE (tem))
                     < GET_MODE_ALIGNMENT (mode))</pre>
                    || (bitpos % GET_MODE_ALIGNMENT (mode) != 0)))
            /* If the type and the field are a constant size and the
               size of the type isn't the same size as the bitfield,
               we must use bitfield operations.
            | | ((bitsize >= 0)
                 && (TREE_CODE (TYPE_SIZE (TREE_TYPE (exp)))
                     == INTEGER_CST)
                 && 0 != compare_tree_int (TYPE_SIZE (TREE_TYPE (exp)),
                                           bitsize)))))
    || (modifier != EXPAND_CONST_ADDRESS
        && modifier != EXPAND_INITIALIZER
        && mode == BLKmode
        && SLOW UNALIGNED ACCESS (mode, alignment)
        && (TYPE_ALIGN (type) > alignment
            || bitpos % TYPE_ALIGN (type) != 0)))
  {
```

### 11.4 Delete garbage

 $\#if\ 0$  blocks that have been there for years, unused functions, unused entire files, dead configurations, dead Makefile logic, dead RTL and tree forms, and on and on and on. Depending on what it is, it may not be obvious if it's garbage or not. Go for the easy ones first.

## 11.5 Use predicates for RTL objects

GCC has simple predicates to see if a given rtx is of some specific class. These predicates simply look at the rtx\_code of the given RTL object and return nonzero if the predicate is true. For example, if an rtx represents a register, then REG\_P (rtx) is nonzero. Unfortunately, lots of code in the middle end and in the back ends does not use these predicates and instead compare the rtx\_code in place: (GET\_CODE (rtx) == REG). Find all the places where such comparisons can be replaced with a predicate. Also, for many common comparisons there is no predicate yet. See which ones are worth having a predicate for, and add them. You can find a number of suggestions in the mailing list archives.

# 12 Security Enhancements

12.1 Return address protection (RAD)

 $\mathrm{Rad}^1$ 

12.2 Repair of control-hijacking attacks (DIRA)

 $Dira^2$ 

12.3 Array bounds checking using segmentation hardware (CASH)

 $Cash^3$ 

12.4 Detecting integer overflows (DIVINE)

 $Divine^4$ 

12.5 Dynamic Information Flow Tracking (GDIF)

 $Gdif^5$ 

<sup>1</sup> http://www.ecsl.cs.sunysb.edu/RAD

<sup>2</sup> http://www.ecsl.cs.sunysb.edu/dira

<sup>3</sup> http://www.ecsl.cs.sunysb.edu/cash

<sup>4</sup> http://www.ecsl.cs.sunysb.edu/iop

<sup>5</sup> http://www.ecsl.cs.sunysb.edu/~lclam/gdif/

# 13 Links

- GCC website<sup>1</sup>
   <sup>2</sup> <sup>3</sup>
- 3. ^4 <sup>5</sup>

http://gcc.gnu.org

<sup>#</sup>ref\_gcc\_summit

http://www.gccsummit.org

<sup>#</sup>ref\_gem\_url

http://research.alexeysmirnov.name/gem

# 14 Contributors

## 15 GNU Free Documentation License

As of July 15, 2009 Wikibooks has moved to a dual-licensing system that supersedes the previous GFDL only licensing. In short, this means that text licensed under the GFDL only can no longer be imported to Wikibooks, retroactive to 1 November 2008. Additionally, Wikibooks text might or might not now be exportable under the GFDL depending on whether or not any content was added and not removed since July 15.

Version 1.3, 3 November 2008 Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.  $<^1>$  Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

#### 15.1 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.

#### 15.2 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

http://fsf.org/

"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.

#### 15.3 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.

#### 15.4 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 networkusing 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.

#### 15.5 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:

- 1. 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.
- 2. 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.
- 3. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- 4. Preserve all the copyright notices of the Document.
- 5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- 6. 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.
- 7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- 8. Include an unaltered copy of this License.
- 9. 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.
- 10. 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.
- 11. 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.
- 12. 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.
- 13. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified version.
- 14. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- 15. 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.

#### 15.6 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".

#### 15.7 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.

# 15.8 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.

#### 15.9 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.

#### **15.10 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.

#### 15.11 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 <sup>2</sup>. 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.

#### 15.12 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.

http://www.gnu.org/copyleft/

# 16 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.

#### Category<sup>1</sup>:

• GNU C Compiler Internals<sup>2</sup>

<sup>1</sup> http://en.wikibooks.org/wiki/Special:Categories

<sup>2</sup> http://en.wikibooks.org/wiki/Category:GNU\_C\_Compiler\_Internals

# 17 Contributors

#### Edits User

- 5 Adrignola<sup>1</sup>
- 1 Darklama<sup>2</sup>
- 1 David $Cary^3$
- 1 Derbeth<sup>4</sup>
- $1 \quad {\rm Dirk} \,\, {\rm H\ddot{u}nniger}^5$
- 1 Jomegat<sup>6</sup>

<sup>1</sup> http://en.wikibooks.org/wiki/User:Adrignola

<sup>2</sup> http://en.wikibooks.org/wiki/User:Darklama

<sup>3</sup> http://en.wikibooks.org/wiki/User:DavidCary

<sup>4</sup> http://en.wikibooks.org/wiki/User:Derbeth

<sup>5</sup> http://en.wikibooks.org/wiki/User:Dirk\_H%25C3%25BCnniger

<sup>6</sup> http://en.wikibooks.org/wiki/User:Jomegat

## List of Figures

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl. html
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/deed.en
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. http://artlibre.org/licence/lal/de
- CFR: Copyright free use.

• EPL: Eclipse Public License. http://www.eclipse.org/org/documents/epl-v10.php

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses<sup>7</sup>. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

<sup>7</sup> Chapter 18 on page 67

1	The people from the Tango! project <sup>8</sup>	
2		
3		
4		
5		
6	Az1568, Webaware	

<sup>8</sup> http://tango.freedesktop.org/The\_People

## 18 Licenses

#### 18.1 GNU GENERAL PUBLIC LICENSE

Version 3 20 June 2007

Copyright © 2007 Free Software Foundation, Inc. <a href="http://fsf.org/">http://fsf.org/</a>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 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 Licens 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

"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 per mission, would make you directly or secondarily liable for infringemen under applicable copyright law, except executing it on a computer o modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and it 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

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 coneve, 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. Conveving 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 apportately 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'. \* e) 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 as 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 notic 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 Conies.

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, are negring 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 whetever licenses to the work the party's predecessor in interest had or could give under the previous pragraph, 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 anyour 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

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the secretice 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 WHE 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 POVED DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SER.

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, INCLEDENTAL, 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 OF THEN PEARITES 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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.

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

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

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 <a href="http://www.gnu.org/licenses/">http://www.gnu.org/licenses/</a>>.

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 -http://www.gnu.org/philosophy/why-not-leph.html>.

#### 18.2 GNU Free Documentation License

Version 1.3. 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <a href="http://fsf.org/">http://fsf.org/</a>

Everyone is permitted to copy and distribute verbatim copies of thi license document, but changing it is not allowed. 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 i world-wide, royalty-free license, unlimited in duration, to use that worl 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 of 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, are 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 DFP designed for human modification. Examples of transparent image formats include PNG, XCF and JFC, 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 Waranty 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. VERREATIM 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 of 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 or the back cover. Both covers must also clearly and legibly dentify 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 rescuerts.

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 networking public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added naterial. 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 notice for your modifications adjacent to the other copyright notice for your modifications adjacent to the other copyright notice for your modifications adjacent to the other copyright 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 is 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, if any given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. \* J. Preserve the Modified Version as stated in the previous sentence for public access to a Transparent copy of the Document, and likewise 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 and the substance and to one of each of the contributor acknowledgements a

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 Over 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 "Delications". You must delete all section 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 electrone 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 Inwariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all 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  ${\tt TERMINATION}$ 

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 http://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 Allike 3.0 license published by Creative Commons Corporation, a not-forprofit 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.

#### 18.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below. 0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

\* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or \* b) under the GAU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

\* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. \* b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the follow-

\* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. \* b) Accompany the Combined Work with a copy of the GNU GPL and this license document. \* c) For a Combined Work that displays copyright notice during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. \* d) Do one of the following: o) Corney the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. ol 1 Use a suitable hardel library mechanism for linking with the Library. A suitable mechanism so one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. \* e) Provide Installation Information under section 6 of the GNU GPL, and only to the extent that such information with a modified version of the Linked Version. (If you use option 4d), the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the followine:

\* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. \* b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser 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 Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.