Valgrind HOWTO

Deepak P.

pdeepak16@vsnl.com

Sandeep S.

sandeep_gect@yahoo.com

24 August 2002

Revision History

Revision 1.1	2002−09−15	Revised by: tab
Converted to XML 4.1.2, added gfdl, reviewed, author revisions		
Revision 1.0	2002−08−24	Revised by: SS
Initial release		

This document is a guide to Valgrind, the `malloc` debugger. Valgrind 1.0.0 is described.

Table of Contents

Table of Contents

1. Background

Dynamic storage allocation plays an important role in C programming; it is also the breeding ground of numerous hard−to−track−down bugs. Freeing an allocated block twice, running off the edge of the malloc'ed buffer, and failing to keep track of addresses of allocated blocks are common errors which frustrate the programmer − debugging them is very difficult due to the errors manifesting themselves as "mysterious behavior" at places far off from the point where the programmer actually committed the blunder.

2. Introduction

2.1. Purpose

Valgrind is an open−source tool for finding memory−management problems in Linux−x86 executables. It detects memory leaks/corruption in the program being run. It is being developed by Julian Seward.

2.2. Acknowledgments

We express our sincere appreciation to Julian Seward for creating Valgrind. Thanks to Mr.Pramode C.E and also friends at the Govt Engineering College, Trichur for their advice and cooperation.

2.3. Copyright and Distribution Policy

2.4. Feedback and Corrections

Kindly forward feedback and criticism to Deepak.P or/and Sandeep.S. We shall be indebted to anybody who points out errors and inaccuracies in this document; we will rectify them as soon as we are informed.

3. Getting it Installed

3.1. Getting Valgrind

Valgrind may be obtained from the following locations:

1. http://developer.kde.org/~sewardj/
2. http://freshmeat.net/projects/valgrind/

3.2. Installing

Uncompress, compile and install it:

```
#tar xvfz valgrind-1.0.0.tar.gz
#cd valgrind-1.0.0
#./configure
#make
#make install
```

Add the path to your `path` variable. Now valgrind is ready to catch the bugs.

4. A Closer View

4.1. Why Valgrind?

As said above, memory management is prone to errors that are too hard to detect. Common errors may be listed as:

1. Use of uninitialized memory
2. Reading/writing memory after it has been freed
3. Reading/writing off the end of malloc'd blocks
4. Reading/writing inappropriate areas on the stack
5. Memory leaks −− where pointers to malloc'd blocks are lost forever
6. Mismatched use of malloc/new/new[] vs free/delete/delete[]
7. Some misuses of the POSIX pthreads API

These errors usually lead to crashes.

This is a situation where we need Valgrind. Valgrind works directly with the executables, with no need to recompile, relink or modify the program to be checked. Valgrind decides whether the program should be modified to avoid memory leak, and also points out the spots of "leak."

Valgrind simulates every single instruction your program executes. For this reason, Valgrind finds errors not only in your application but also in all supporting dynamically−linked (.so−format) libraries, including the GNU C library, the X client libraries, Qt if you work with KDE, and so on. That often includes libraries, for example the GNU C library, which may contain memory access violations.

4.2. Usage

4.2.1. Invoking Valgrind

The checking may be performed by simply placing the word **valgrind** just before the normal command used to invoke the program. For example:

```
#valgrind ps −ax
```

Valgrind provides thousands of options. We deliberately avoid them, not to make this article boring.

The output contains the usual output of **ps −ax** also with the detailed report by valgrind. Any error (memory related) is pointed out in the error report.

4.2.2. How to Identify the Error from the Error Report

Consider the output of Valgrind for some test program:

```
   ==1353== Invalid read of size 4
   ==1353==    at 0x80484F6: print (valg_eg.c:7)
   ==1353==    by 0x8048561: main (valg_eg.c:16)
   ==1353==    by 0x4026D177: __libc_start_main
(../sysdeps/generic/libc-start.c   :129)
```

```
   ==1353==    by 0x80483F1: free@@GLIBC_2.0 (in /home/deepu/valg/a.out)
   ==1353==    Address 0x40C9104C is 0 bytes after a block of size 40
alloc'd
   ==1353==    at 0x40046824: malloc (vg_clientfuncs.c:100)
   ==1353==    by 0x8048524: main (valg_eg.c:12)
   ==1353==    by 0x4026D177: __libc_start_main
(../sysdeps/generic/libc-start.c   :129)
   ==1353==    by 0x80483F1: free@@GLIBC_2.0 (in /home/deepu/valg/a.out)
```

Here, 1353 is the process ID. This part of the error report says that a read error has occurred at line number 7, in the function `print`. The function `print` is called by function `main`, and both are in the file `valg_eg.c`. The function `main` is called by the function `__libc_start_main` at line number 129, in `../sysdeps/generic/libc-start.c`. The function `__libc_start_main` is called by `free@@GLIBC_2.0` in the file `/home/deepu/valg/a.out.` Similarly details of calling `malloc` are also given.

4.2.3. Types of Errors with Examples

Valgrind can only really detect two types of errors: use of illegal address and use of undefined values. Nevertheless, this is enough to discover all sorts of memory management problems in a program. Some common errors are given below.

4.2.3.1. Use of uninitialized memory

Sources of uninitialized data are:

- local variables that have not been initialized.
- The contents of malloc'd blocks, before writing something there.

This is not a problem with `calloc` since it initializes each allocated bytes with 0. The `new` operator in C++ is similar to `malloc`. Fields of the created object will be uninitialized.

Sample program:

```
#include <stdlib.h>
int main()
{
        int p, t;
        if (p == 5)             /*Error occurs here*/
                t = p+1;
        return 0;
}
```

Here the value of `p` is uninitialized, therefore `p` may contain some random value (garbage), so an error may occur at the condition check. An uninitialized variable will cause error in 2 situations:

- When it is used to determine the outcome of a conditional branch. Eg:'if (p == 5)' in the above program.
- When it is used to generate a memory address. Eg: In the above program let there be an integer array a[10], and if you write 'a[p] = 1', it will generate an error.

4.2.3.2. Illegal read/write

Illegal read/write errors occurs when you try to read/write from/to an address that is not in the address range of your program.

Sample program:

```
#include <stdlib.h>
int main()
{
        int *p, i, a;
        p = malloc(10*sizeof(int));
        p[11] = 1;                /* invalid write error */
        a = p[11];                /* invalid read error */
                free(p);
        return 0;
}
```

Here you are trying to read/write from/to address (p+sizeof(int)*11) which is not allocated to the program.

4.2.3.3. Invalid free

Valgrind keeps track of blocks allocated to your program with `malloc/new`. So it can easily check whether argument to free/delete is valid or not.

Sample program:

```
#include <stdlib.h>
int main()
{
        int *p, i;
        p = malloc(10*sizeof(int));
        for(i = 0;i < 10;i++)
                p[i] = i;
        free(p);
        free(p);        /* Error: p has already been freed */
        return 0;
}
```

Valgrind checks the address, which is given as argument to free. If it is an address that has already been freed you will be told that the free is invalid.

4.2.3.4. Mismatched Use of Functions

In C++ you can allocate and free memory using more than one function, but the following rules must be followed:

- If allocated with `malloc`, `calloc`, `realloc`, `valloc` or `memalign`, you must deallocate with `free`.
- If allocated with `new[]`, you must deallocate with `delete[]`.
- If allocated with `new`, you must deallocate with `delete`.

Sample program:

```
#include <stdlib.h>
int main()
{
        int *p, i;
        p = ( int* ) malloc(10*sizeof(int));
        for(i = 0;i < 10;i++)
                p[i] = i;
        delete(p);                /* Error: function mismatch */
        return 0;
}
```

Output by valgrind is:

```
             ==1066== ERROR SUMMARY: 1 errors from 1 contexts (suppressed:
0 from 0)
             ==1066== malloc/free: in use at exit: 0 bytes in 0 blocks.
             ==1066== malloc/free: 1 allocs, 1 frees, 40 bytes allocated.
             ==1066== For a detailed leak analysis,  rerun with:
--leak-check=yes
             ==1066== For counts of detected errors, rerun with: -v
```

>From the above "ERROR SUMMARY" it is clear that there is 0 bytes in 0 blocks in use at exit, which means that the malloc'd have been freed by `delete`. Therefore this is not a problem in Linux, but this program may crash on some other platform.

4.2.3.5. Errors Occur Due to Invalid System Call Parameter

Valgrind checks all parameters to system calls.

Sample program:

```
#include <stdlib.h>
#include <unistd.h>
int main()
{
        int *p;
        p = malloc(10);
        read(0, p, 100);        /* Error: unaddressable bytes */
        free(p);
        return 0;
}
```

```
             ==1045== Syscall param read(buf) contains unaddressable
byte(s)
             ==1045==    at 0x4032AF44: __libc_read (in
/lib/i686/libc-2.2.2.so)
             ==1045==    by 0x4026D177: __libc_start_main
(../sysdeps/generic/libc-start.c:129)
             ==1045==    by 0x80483E1: read@@GLIBC_2.0 (in
/home/deepu/valg/a.out)
```

Here, `buf = p` contains the address of a 10 byte block. The `read` system call tries to read 100 bytes from standard input and place it at `p`. But the bytes after the first 10 are unaddressable.

4.2.3.6. Memory Leak Detection

Consider the following program:

```
#include <stdlib.h>
int main()
{
        int *p, i;
        p = malloc(5*sizeof(int));
        for(i = 0;i < 5;i++)
                p[i] = i;
        return 0;
}
```

```
             ==1048== LEAK SUMMARY:
             ==1048==    definitely lost: 20 bytes in 1 blocks.
             ==1048==    possibly lost:   0 bytes in 0 blocks.
             ==1048==    still reachable: 0 bytes in 0 blocks.
```

In the above program `p` contains the address of a 20−byte block. But it is not freed anywhere in the program. So the pointer to this 20 byte block is lost forever. This is known as memory leaking. We can get the leak summary by using the Valgrind option `--leak-check=yes`.

4.2.4. How to Suppress Errors

Valgrind detects numerous problems in many programs which come pre−installed on your GNU/Linux system. You can't easily fix these, but you don't want to see these errors (and yes, there are many!). So Valgrind reads a list of errors to suppress at startup, from a suppression file ending in `.supp`.

Suppression files may be modified. This is useful if part of your project contains errors you can't or don't want to fix, yet you don't want to continuously be reminded of them. The format of the file is as follows.

```
{
             Error name
             Type
             fun:function name, which contains the error to suppress
        fun:function name, which calls the function specified above
}
```

```
Error name can be any name.
             type=ValueN, if the error is an uninitialized value error.
                 =AddrN, if it is an address error.(N=sizeof(data type))
                 =Free, if it is a free error (eg:mismatched free)
                 =Cond, if error is due to uninitialized CPU condition code.
                 =Param, if it is an invalid system call parameter error.
```

You can then run the program with:

```
valgrind --suppressions=path/to/the/supp_file.supp testprog
```

The output will not contain the errors specified in the suppression file.

4.3. Limitations and Dependencies of Valgrind.

No software is free from limitations. The same is the case of Valgrind, however most programs work fine. The limitations are listed below.

1. Program runs 25 to 50 times slower.
2. Increased memory consumption.
3. Highly optimized code (compiled with −O1, −O2 options) may sometimes cheat Valgrind.
4. Valgrind relies on dynamic linking mechanism.

Valgrind is closely tied to details of the CPU, operating system and to a less extent, compiler and basic C libraries. Presently Valgrind works only on the Linux platform (kernels 2.2.X or 2.4.X) on x86s. Glibc 2.1.X or 2.2.X is also required for Valgrind.

---

5. Let's Go Deeper

Valgrind simulates an Intel x86 processor and runs our test program in this synthetic processor. The two processors are not exactly same. Valgrind is compiled into a shared object, valgrind.so. A shell script `valgrind` sets the `LD_PRELOAD` environment variable to point to valgrind.so. This causes the .so to be loaded as an extra library to any subsequently executed dynamically−linked ELF binary, permitting the program to be debugged.

The dynamic linker calls the initialization function of Valgrind. Then the synthetic CPU takes control from the real CPU. In the memory there may be some other .so files. The dynamic linker calls the initialization function of all such .so files. Now the dynamic linker calls the `main` of the loaded program. When main returns, the synthetic CPU calls the finalization function of valgrind.so. During the execution of the finalization function, summary of all errors detected are printed and memory leaks are checked. Finalization function exits giving back the control from the synthetic CPU to the real one.

5.1. How Valgrind Tracks Validity of Each Byte

For every byte processed, the synthetic processor maintains 9 bits, 8 'V' bits and 1 'A' bit. The 'V' bits indicate the validity of the 8 bits in the byte and the 'A' bit indicates validity of the byte address. These valid−value(V) bits are checked only in two situations:

1. when data is used for address generation,
2. when control flow decision is to be made.

In any of these two situations, if the data is found to be undefined an error report will be generated. But no error reports are generated while copying or adding undefined data.

However the case with floating−point data is different. During a floating−point read instruction the 'V' bits corresponding to the data are checked. Thus copying of uninitialized value will produce error in case of floating−point numbers.

```
#include <stdlib.h>
int main()
{
        int *p, *a;
        p = malloc(10*sizeof(int));
        a = malloc(10*sizeof(int));
        a[3] = p[3];
        free(a);
        free(p);
        return 0;
}

/*  produce no errors */
```

```
#include <stdlib.h>
int main()
{
        float *p, *a;
        p = malloc(10*sizeof(float));
        a = malloc(10*sizeof(float));
        a[3] = p[3];
```

```
        free(a);
        free(p);
        return 0;
}

/* produces error */
```

All bytes that are in memory but not in CPU have an associated valid−address(A) bit, which indicates whether the corresponding memory location is accessible by the program. When a program starts, the 'A' bits corresponding to each global variables are set. When a call `malloc`, `new` or any other memory allocating function is made, the 'A' bits corresponding to the allocated bytes are set. Upon freeing the allocated block using `free/new/new` the corresponding 'A' bits are cleared. While doing a system call the 'A' bits are changed appropriately.

When values are loaded from memory the 'A' bits corresponding to each bytes are checked by Valgrind, and if the 'A' bit corresponding to a byte is set then its 'V' bits is checked. If the 'V' bits are not set, an error will be generated and the 'V' bits are set to indicate validity. This avoids long chain of errors. If the 'A' bit corresponding to a loaded byte is 0 then its 'V' bits are forced to set, despite the value being invalid.

Have a look on the following program. Run it.

```
#include <stdlib.h>
int main()
{
        int *p, j;
        p = malloc(5*sizeof(int));
        j = p[5];
        if (p[5] == 1)
                i = p[5]+1;
        free(p);
        return 0;
}
```

Here two errors occur. Both of them are due to the accessing address location `p + sizeof(int)*5` which is not allocated to the program. During the execution of `j = p[5]`, since the address `p + sizeof(int)*5` is invalid, the 'V' bits of 4 bytes starting at location `p+sizeof(int)*5` are forced to set. Therefore uninitialized value occurs neither during the execution of `j = p[5]` nor during the execution of `if(p[5]==1)`.

5.2. Cache Profiling

Modern x86 machines use two levels of caching. These levels are L1 and L2, in which L1 is a split cache that consists of Instruction cache(I1) and Data cache(D1). L2 is a unified cache.

The configuration of a cache means its size, associativity and number of lines. If the data requested by the processor appears in the upper level it is called a hit. If the data is not found in the upper level, the request is called a miss. The lower level in the hierarchy is then accessed to retrieve the block containing requested data. In modern machines L1 is first searched for data/instruction requested by the processor. If it is a hit then that data/instruction is copied to some register in the processor. Otherwise L2 is searched. If it is a hit then data/instruction is copied to L1 and from there it is copied to a register. If the request to L2 also is a miss then main memory has to be accessed.

Valgrind can simulate the cache, meaning it can display the things that occur in the cache when a program is running. For this, first compile your program with `-g` option as usual. Then use the shell script `cachegrind` instead of `valgrind`.

Sample output:

```
==7436== I1  refs:      12,841
==7436== I1  misses:       238
==7436== L2i misses:       237
==7436== I1  miss rate:   1.85%
==7436== L2i miss rate:   1.84%
==7436==
==7436== D   refs:       5,914  (4,626 rd + 1,288 wr)
==7436== D1  misses:       357  (  324 rd +    33 wr)
==7436== L2d misses:       352  (  319 rd +    33 wr)
==7436== D1  miss rate:    6.0% (  7.0%   +   2.5%  )
==7436== L2d miss rate:    5.9% (  6.8%   +   2.5%  )
==7436==
==7436== L2 refs:          595  (  562 rd +    33 wr)
==7436== L2 misses:        589  (  556 rd +    33 wr)
==7436== L2 miss rate:     3.1% (  3.1%   +   2.5%  )
```

```
   L2i misses means the number of instruction misses that occur in L2
cache.
   L2d misses means the number of data misses that occur in L2 cache.
   Total number of data references = Number of reads + Number of writes.
   Miss rate means fraction of misses that are not found in the upper
level.
```

The shell script `cachegrind` also produces a file, `cachegrind.out`, that contains line−by−line cache profiling information which is not humanly understandable. A program `vg_annotate` can easily interpret this information. If the shell script `vg_annotate` is used without any arguments it will read the file `cachegrind.out` and produce an output which is humanly understandable.

When C, C++ or assembly source programs are passed as input to `vg_annotate` it displays the number of cache reads, writes, misses etc.

```
I1 cache:         16384 B, 32 B, 4-way associative
D1 cache:         16384 B, 32 B, 4-way associative
L2 cache:         262144 B, 32 B, 8-way associative
Command:          ./a.out
Events recorded:  Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw
Events shown:     Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw
Event sort order: Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw
Thresholds:       99 0 0 0 0 0 0 0 0
Include dirs:
User annotated:   valg_flo.c
Auto-annotation:  off
```

User−annotated source: `valg_flo.c`:

```
Ir I1mr I2mr Dr D1mr D2mr Dw D1mw D2mw

 .   .    .   .    .    .   .    .    .   #include<stdlib.h>
 .   .    .   .    .    .   .    .    .   int main()
 3   1    1   .    .    .   1    0    0   {
 .   .    .   .    .    .   .    .    .           float *p, *a;
```

```
6   1   1   .   .    .   3   0    0           p = malloc(10*sizeof(float));
6   0   0   .   .    .   3   0    0           a = malloc(10*sizeof(float));
6   1   1   3   1    1   1   1    1           a[3] = p[3];
4   0   0   1   0    0   1   0    0           free(a);
4   0   0   1   0    0   1   0    0           free(p);
2   0   0   2   0    0   .   .    .   }
```

- Ir = Total instruction cache reads.
- I1mr = I1 cache read misses.
- I2mr = L2 cache instruction read misses.

6. Concluding Remarks

This document has gone through the basics of Valgrind. Once you understand the basic concept it is not difficult to make steps on your own.

If you have found any glaring typos, or outdated info in this document, please let us know.

7. References

1. http://developer.kde.org/~sewardj/docs/
2. The most valuable source of information is the source code itself.

A. GNU Free Documentation License

Version 1.1, March 2000

A.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written 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.

A.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

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. (For example, 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.

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 "Transparent" copy of the Document means a machine−readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and 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 has been designed to thwart or discourage subsequent modification by readers is not Transparent. 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 designed for human modification. Opaque formats include PostScript, PDF, 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 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.

A.3. 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.

A.4. COPYING IN QUANTITY

If you publish printed copies 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 publicly−accessible computer−network location containing a complete Transparent copy of the Document, free of added material, which the general network−using public has access to download anonymously at no charge using public−standard network protocols. 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.

A.5. 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 less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

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.

A.6. COMBINING DOCUMENTS

A.7. 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.

A.8. 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, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self–contained works thus compiled with the Document, on account of their being thus compiled, if they 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 quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.9. 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 provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11. 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.

A.12. 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 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front−Cover Texts being LIST, and with the Back−Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front−Cover Texts, write "no Front−Cover Texts" instead of "Front−Cover Texts being LIST"; likewise for Back−Cover Texts.

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.