User's Guide to gperf 3.0.1

The GNU Perfect Hash Function Generator
Edition 3.0.1, 12 June 2003

Douglas C. Schmidt
Bruno Haible

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. 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.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

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

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program 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.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you

indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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 convey 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 an 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 2
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, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.
```

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

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

```
Gnomovision version 69, Copyright (C) year  name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This 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 Library General Public License instead of this License.

Contributors to GNU gperf Utility

- The GNU `gperf` perfect hash function generator utility was written in GNU C++ by Douglas C. Schmidt. The general idea for the perfect hash function generator was inspired by Keith Bostic's algorithm written in C, and distributed to net.sources around 1984. The current program is a heavily modified, enhanced, and extended implementation of Keith's basic idea, created at the University of California, Irvine. Bugs, patches, and suggestions should be reported to `<bug-gnu-gperf@gnu.org>`.
- Special thanks is extended to Michael Tiemann and Doug Lea, for providing a useful compiler, and for giving me a forum to exhibit my creation.

  In addition, Adam de Boor and Nels Olson provided many tips and insights that greatly helped improve the quality and functionality of `gperf`.
- Bruno Haible enhanced and optimized the search algorithm. He also rewrote the input routines and the output routines for better reliability, and added a testsuite.

1 Introduction

`gperf` is a perfect hash function generator written in C++. It transforms an *n* element user-specified keyword set *W* into a perfect hash function *F*. *F* uniquely maps keywords in *W* onto the range 0..*k*, where *k* >= *n-1*. If *k* = *n-1* then *F* is a *minimal* perfect hash function. `gperf` generates a 0..*k* element static lookup table and a pair of C functions. These functions determine whether a given character string *s* occurs in *W*, using at most one probe into the lookup table.

`gperf` currently generates the reserved keyword recognizer for lexical analyzers in several production and research compilers and language processing tools, including GNU C, GNU C++, GNU Java, GNU Pascal, GNU Modula 3, and GNU indent. Complete C++ source code for `gperf` is available from `http://ftp.gnu.org/pub/gnu/gperf/`. A paper describing `gperf`'s design and implementation in greater detail is available in the Second USENIX C++ Conference proceedings or from `http://www.cs.wustl.edu/~schmidt/resume.html`.

2 Static search structures and GNU `gperf`

A *static search structure* is an Abstract Data Type with certain fundamental operations, e.g., *initialize*, *insert*, and *retrieve*. Conceptually, all insertions occur before any retrievals. In practice, `gperf` generates a *static* array containing search set keywords and any associated attributes specified by the user. Thus, there is essentially no execution-time cost for the insertions. It is a useful data structure for representing *static search sets*. Static search sets occur frequently in software system applications. Typical static search sets include compiler reserved words, assembler instruction opcodes, and built-in shell interpreter commands. Search set members, called *keywords*, are inserted into the structure only once, usually during program initialization, and are not generally modified at run-time.

Numerous static search structure implementations exist, e.g., arrays, linked lists, binary search trees, digital search tries, and hash tables. Different approaches offer trade-offs between space utilization and search time efficiency. For example, an n element sorted array is space efficient, though the average-case time complexity for retrieval operations using binary search is proportional to log n. Conversely, hash table implementations often locate a table entry in constant time, but typically impose additional memory overhead and exhibit poor worst case performance.

Minimal perfect hash functions provide an optimal solution for a particular class of static search sets. A minimal perfect hash function is defined by two properties:

- It allows keyword recognition in a static search set using at most *one* probe into the hash table. This represents the "perfect" property.
- The actual memory allocated to store the keywords is precisely large enough for the keyword set, and *no larger*. This is the "minimal" property.

For most applications it is far easier to generate *perfect* hash functions than *minimal perfect* hash functions. Moreover, non-minimal perfect hash functions frequently execute faster than minimal ones in practice. This phenomena occurs since searching a sparse keyword table increases the probability of locating a "null" entry, thereby reducing string comparisons. `gperf`'s default behavior generates *near-minimal* perfect hash functions for keyword sets. However, `gperf` provides many options that permit user control over the degree of minimality and perfection.

Static search sets often exhibit relative stability over time. For example, Ada's 63 reserved words have remained constant for nearly a decade. It is therefore frequently worthwhile to expend concerted effort building an optimal search structure *once*, if it subsequently receives heavy use multiple times. `gperf` removes the drudgery associated with constructing time- and space-efficient search structures by hand. It has proven a useful and practical tool for serious programming projects. Output from `gperf` is currently used in several production and research compilers, including GNU C, GNU C++, GNU Java, GNU Pascal, and GNU Modula 3. The latter two compilers are not yet part of the official GNU distribution. Each compiler utilizes `gperf` to automatically generate static search structures that efficiently identify their respective reserved keywords.

3 High-Level Description of GNU gperf

The perfect hash function generator `gperf` reads a set of "keywords" from an input file (or from the standard input by default). It attempts to derive a perfect hashing function that recognizes a member of the *static keyword set* with at most a single probe into the lookup table. If `gperf` succeeds in generating such a function it produces a pair of C source code routines that perform hashing and table lookup recognition. All generated C code is directed to the standard output. Command-line options described below allow you to modify the input and output format to `gperf`.

By default, `gperf` attempts to produce time-efficient code, with less emphasis on efficient space utilization. However, several options exist that permit trading-off execution time for storage space and vice versa. In particular, expanding the generated table size produces a sparse search structure, generally yielding faster searches. Conversely, you can direct `gperf` to utilize a C `switch` statement scheme that minimizes data space storage size. Furthermore, using a C `switch` may actually speed up the keyword retrieval time somewhat. Actual results depend on your C compiler, of course.

In general, `gperf` assigns values to the bytes it is using for hashing until some set of values gives each keyword a unique value. A helpful heuristic is that the larger the hash value range, the easier it is for `gperf` to find and generate a perfect hash function. Experimentation is the key to getting the most from `gperf`.

3.1 Input Format to gperf

You can control the input file format by varying certain command-line arguments, in particular the '`-t`' option. The input's appearance is similar to GNU utilities `flex` and `bison` (or UNIX utilities `lex` and `yacc`). Here's an outline of the general format:

```
declarations
%%
keywords
%%
functions
```

Unlike `flex` or `bison`, the declarations section and the functions section are optional. The following sections describe the input format for each section.

It is possible to omit the declaration section entirely, if the '`-t`' option is not given. In this case the input file begins directly with the first keyword line, e.g.:

```
january
february
march
april
...
```

3.1.1 Declarations

The keyword input file optionally contains a section for including arbitrary C declarations and definitions, `gperf` declarations that act like command-line options, as well as for providing a user-supplied `struct`.

3.1.1.1 User-supplied struct

If the '-t' option (or, equivalently, the '%struct-type' declaration) *is* enabled, you *must* provide a C struct as the last component in the declaration section from the input file. The first field in this struct must be of type char * or const char * if the '-P' option is not given, or of type int if the option '-P' (or, equivalently, the '%pic' declaration) is enabled. This first field must be called 'name', although it is possible to modify its name with the '-K' option (or, equivalently, the '%define slot-name' declaration) described below.

Here is a simple example, using months of the year and their attributes as input:

```
struct month { char *name; int number; int days; int leap_days; };
%%
january,   1, 31, 31
february,  2, 28, 29
march,     3, 31, 31
april,     4, 30, 30
may,       5, 31, 31
june,      6, 30, 30
july,      7, 31, 31
august,    8, 31, 31
september, 9, 30, 30
october,  10, 31, 31
november, 11, 30, 30
december, 12, 31, 31
```

Separating the struct declaration from the list of keywords and other fields are a pair of consecutive percent signs, '%%', appearing left justified in the first column, as in the UNIX utility lex.

If the struct has already been declared in an include file, it can be mentioned in an abbreviated form, like this:

```
struct month;
%%
january,   1, 31, 31
...
```

3.1.1.2 Gperf Declarations

The declaration section can contain gperf declarations. They influence the way gperf works, like command line options do. In fact, every such declaration is equivalent to a command line option. There are three forms of declarations:

1. Declarations without argument, like '%compare-lengths'.
2. Declarations with an argument, like '%switch=*count*'.
3. Declarations of names of entities in the output file, like '%define lookup-function-name *name*'.

When a declaration is given both in the input file and as a command line option, the command-line option's value prevails.

The following gperf declarations are available.

'%delimiters=*delimiter-list*'
: Allows you to provide a string containing delimiters used to separate keywords from their attributes. The default is ",". This option is essential if you want to use keywords that have embedded commas or newlines.

'%struct-type'
: Allows you to include a `struct` type declaration for generated code; see above for an example.

'%ignore-case'
: Consider upper and lower case ASCII characters as equivalent. The string comparison will use a case insignificant character comparison. Note that locale dependent case mappings are ignored.

'%language=*language-name*'
: Instructs `gperf` to generate code in the language specified by the option's argument. Languages handled are currently:

  'KR-C' Old-style K&R C. This language is understood by old-style C compilers and ANSI C compilers, but ANSI C compilers may flag warnings (or even errors) because of lacking '`const`'.

  'C' Common C. This language is understood by ANSI C compilers, and also by old-style C compilers, provided that you `#define const` to empty for compilers which don't know about this keyword.

  'ANSI-C' ANSI C. This language is understood by ANSI C compilers and C++ compilers.

  'C++' C++. This language is understood by C++ compilers.

  The default is C.

'%define slot-name *name*'
: This declaration is only useful when option '`-t`' (or, equivalently, the '`%struct-type`' declaration) has been given. By default, the program assumes the structure component identifier for the keyword is '`name`'. This option allows an arbitrary choice of identifier for this component, although it still must occur as the first field in your supplied `struct`.

'%define initializer-suffix *initializers*'
: This declaration is only useful when option '`-t`' (or, equivalently, the '`%struct-type`' declaration) has been given. It permits to specify initializers for the structure members following *slot-name* in empty hash table entries. The list of initializers should start with a comma. By default, the emitted code will zero-initialize structure members following *slot-name*.

'%define hash-function-name *name*'
: Allows you to specify the name for the generated hash function. Default name is '`hash`'. This option permits the use of two hash tables in the same file.

'%define lookup-function-name *name*'
: Allows you to specify the name for the generated lookup function. Default name is '`in_word_set`'. This option permits multiple generated hash functions to be used in the same application.

'%define class-name *name*'
: This option is only useful when option '-L C++' (or, equivalently, the '%language=C++' declaration) has been given. It allows you to specify the name of generated C++ class. Default name is Perfect_Hash.

'%7bit'
: This option specifies that all strings that will be passed as arguments to the generated hash function and the generated lookup function will solely consist of 7-bit ASCII characters (bytes in the range 0..127). (Note that the ANSI C functions isalnum and isgraph do *not* guarantee that a byte is in this range. Only an explicit test like 'c >= 'A' && c <= 'Z'' guarantees this.)

'%compare-lengths'
: Compare keyword lengths before trying a string comparison. This option is mandatory for binary comparisons (see Section 3.3 [Binary Strings], page 18). It also might cut down on the number of string comparisons made during the lookup, since keywords with different lengths are never compared via strcmp. However, using '%compare-lengths' might greatly increase the size of the generated C code if the lookup table range is large (which implies that the switch option '-S' or '%switch' is not enabled), since the length table contains as many elements as there are entries in the lookup table.

'%compare-strncmp'
: Generates C code that uses the strncmp function to perform string comparisons. The default action is to use strcmp.

'%readonly-tables'
: Makes the contents of all generated lookup tables constant, i.e., "readonly". Many compilers can generate more efficient code for this by putting the tables in readonly memory.

'%enum'
: Define constant values using an enum local to the lookup function rather than with #defines. This also means that different lookup functions can reside in the same file. Thanks to James Clark <jjc@ai.mit.edu>.

'%includes'
: Include the necessary system include file, <string.h>, at the beginning of the code. By default, this is not done; the user must include this header file himself to allow compilation of the code.

'%global-table'
: Generate the static table of keywords as a static global variable, rather than hiding it inside of the lookup function (which is the default behavior).

'%pic'
: Optimize the generated table for inclusion in shared libraries. This reduces the startup time of programs using a shared library containing the generated code. If the '%struct-type' declaration (or, equivalently, the option '-t') is also given, the first field of the user-defined struct must be of type 'int', not 'char *', because it will contain offsets into the string pool instead of actual strings. To convert such an offset to a string, you can use the expression 'stringpool + *o*', where *o* is the offset. The string pool name can be changed through the '%define string-pool-name' declaration.

'%define string-pool-name *name*'
: Allows you to specify the name of the generated string pool created by the declaration '%pic' (or, equivalently, the option '-P'). The default name is 'stringpool'. This declaration permits the use of two hash tables in the same file, with '%pic' and even when the '%global-table' declaration (or, equivalently, the option '-G') is given.

'%null-strings'
: Use NULL strings instead of empty strings for empty keyword table entries. This reduces the startup time of programs using a shared library containing the generated code (but not as much as the declaration '%pic'), at the expense of one more test-and-branch instruction at run time.

'%define word-array-name *name*'
: Allows you to specify the name for the generated array containing the hash table. Default name is 'wordlist'. This option permits the use of two hash tables in the same file, even when the option '-G' (or, equivalently, the '%global-table' declaration) is given.

'%switch=*count*'
: Causes the generated C code to use a switch statement scheme, rather than an array lookup table. This can lead to a reduction in both time and space requirements for some input files. The argument to this option determines how many switch statements are generated. A value of 1 generates 1 switch containing all the elements, a value of 2 generates 2 tables with 1/2 the elements in each switch, etc. This is useful since many C compilers cannot correctly generate code for large switch statements. This option was inspired in part by Keith Bostic's original C program.

'%omit-struct-type'
: Prevents the transfer of the type declaration to the output file. Use this option if the type is already defined elsewhere.

3.1.1.3 C Code Inclusion

Using a syntax similar to GNU utilities flex and bison, it is possible to directly include C source text and comments verbatim into the generated output file. This is accomplished by enclosing the region inside left-justified surrounding '%{', '%}' pairs. Here is an input fragment based on the previous example that illustrates this feature:

```
%{
#include <assert.h>
/* This section of code is inserted directly into the output. */
int return_month_days (struct month *months, int is_leap_year);
%}
struct month { char *name; int number; int days; int leap_days; };
%%
january,   1, 31, 31
february,  2, 28, 29
march,     3, 31, 31
...
```

3.1.2 Format for Keyword Entries

The second input file format section contains lines of keywords and any associated attributes you might supply. A line beginning with '#' in the first column is considered a comment. Everything following the '#' is ignored, up to and including the following newline. A line beginning with '%' in the first column is an option declaration and must not occur within the keywords section.

The first field of each non-comment line is always the keyword itself. It can be given in two ways: as a simple name, i.e., without surrounding string quotation marks, or as a string enclosed in double-quotes, in C syntax, possibly with backslash escapes like `\"` or `\234` or `\xa8`. In either case, it must start right at the beginning of the line, without leading whitespace. In this context, a "field" is considered to extend up to, but not include, the first blank, comma, or newline. Here is a simple example taken from a partial list of C reserved words:

```
# These are a few C reserved words, see the c.gperf file
# for a complete list of ANSI C reserved words.
unsigned
sizeof
switch
signed
if
default
for
while
return
```

Note that unlike `flex` or `bison` the first '%%' marker may be elided if the declaration section is empty.

Additional fields may optionally follow the leading keyword. Fields should be separated by commas, and terminate at the end of line. What these fields mean is entirely up to you; they are used to initialize the elements of the user-defined `struct` provided by you in the declaration section. If the '-t' option (or, equivalently, the '%struct-type' declaration) is *not* enabled these fields are simply ignored. All previous examples except the last one contain keyword attributes.

3.1.3 Including Additional C Functions

The optional third section also corresponds closely with conventions found in `flex` and `bison`. All text in this section, starting at the final '%%' and extending to the end of the input file, is included verbatim into the generated output file. Naturally, it is your responsibility to ensure that the code contained in this section is valid C.

3.1.4 Where to place directives for GNU indent.

If you want to invoke GNU `indent` on a `gperf` input file, you will see that GNU `indent` doesn't understand the '%%', '%{' and '%}' directives that control `gperf`'s interpretation of the input file. Therefore you have to insert some directives for GNU `indent`. More precisely, assuming the most general input file structure

```
declarations part 1
%{
verbatim code
%}
declarations part 2
%%
keywords
%%
functions
```

you would insert '*INDENT-OFF*' and '*INDENT-ON*' comments as follows:

```
/* *INDENT-OFF* */
declarations part 1
%{
/* *INDENT-ON* */
verbatim code
/* *INDENT-OFF* */
%}
declarations part 2
%%
keywords
%%
/* *INDENT-ON* */
functions
```

3.2 Output Format for Generated C Code with gperf

Several options control how the generated C code appears on the standard output. Two C function are generated. They are called `hash` and `in_word_set`, although you may modify their names with a command-line option. Both functions require two arguments, a string, `char *` *str*, and a length parameter, `int` *len*. Their default function prototypes are as follows:

`unsigned int` **hash** (`const char *` *str*, `unsigned int` *len*) Function

By default, the generated `hash` function returns an integer value created by adding *len* to several user-specified *str* byte positions indexed into an *associated values* table stored in a local static array. The associated values table is constructed internally by `gperf` and later output as a static local C array called '`hash_table`'. The relevant selected positions (i.e. indices into *str*) are specified via the '`-k`' option when running `gperf`, as detailed in the *Options* section below (see Chapter 4 [Options], page 19).

in_word_set (`const char *` *str*, `unsigned int` *len*) Function

If *str* is in the keyword set, returns a pointer to that keyword. More exactly, if the option '`-t`' (or, equivalently, the '`%struct-type`' declaration) was given, it returns a pointer to the matching keyword's structure. Otherwise it returns `NULL`.

If the option '`-c`' (or, equivalently, the '`%compare-strncmp`' declaration) is not used, *str* must be a NUL terminated string of exactly length *len*. If '`-c`' (or, equivalently, the

'%compare-strncmp' declaration) is used, *str* must simply be an array of *len* bytes and does not need to be NUL terminated.

The code generated for these two functions is affected by the following options:

'-t'
'--struct-type'
: Make use of the user-defined `struct`.

'-S *total-switch-statements*'
'--switch=*total-switch-statements*'
: Generate 1 or more C `switch` statement rather than use a large, (and potentially sparse) static array. Although the exact time and space savings of this approach vary according to your C compiler's degree of optimization, this method often results in smaller and faster code.

If the '-t' and '-S' options (or, equivalently, the '%struct-type' and '%switch' declarations) are omitted, the default action is to generate a `char *` array containing the keywords, together with additional empty strings used for padding the array. By experimenting with the various input and output options, and timing the resulting C code, you can determine the best option choices for different keyword set characteristics.

3.3 Use of NUL bytes

By default, the code generated by `gperf` operates on zero terminated strings, the usual representation of strings in C. This means that the keywords in the input file must not contain NUL bytes, and the *str* argument passed to `hash` or `in_word_set` must be NUL terminated and have exactly length *len*.

If option '-c' (or, equivalently, the '%compare-strncmp' declaration) is used, then the *str* argument does not need to be NUL terminated. The code generated by `gperf` will only access the first *len*, not *len+1*, bytes starting at *str*. However, the keywords in the input file still must not contain NUL bytes.

If option '-l' (or, equivalently, the '%compare-lengths' declaration) is used, then the hash table performs binary comparison. The keywords in the input file may contain NUL bytes, written in string syntax as `\000` or `\x00`, and the code generated by `gperf` will treat NUL like any other byte. Also, in this case the '-c' option (or, equivalently, the '%compare-strncmp' declaration) is ignored.

4 Invoking gperf

There are *many* options to `gperf`. They were added to make the program more convenient for use with real applications. "On-line" help is readily available via the '`--help`' option. Here is the complete list of options.

4.1 Specifying the Location of the Output File

'`--output-file=`*file*'
: Allows you to specify the name of the file to which the output is written to.

The results are written to standard output if no output file is specified or if it is '`-`'.

4.2 Options that affect Interpretation of the Input File

These options are also available as declarations in the input file (see Section 3.1.1.2 [Gperf Declarations], page 12).

'`-e` *keyword-delimiter-list*'
'`--delimiters=`*keyword-delimiter-list*'
: Allows you to provide a string containing delimiters used to separate keywords from their attributes. The default is `","`. This option is essential if you want to use keywords that have embedded commas or newlines. One useful trick is to use -e'TAB', where TAB is the literal tab character.

'`-t`'
'`--struct-type`'
: Allows you to include a `struct` type declaration for generated code. Any text before a pair of consecutive '`%%`' is considered part of the type declaration. Keywords and additional fields may follow this, one group of fields per line. A set of examples for generating perfect hash tables and functions for Ada, C, C++, Pascal, Modula 2, Modula 3 and JavaScript reserved words are distributed with this release.

'`--ignore-case`'
: Consider upper and lower case ASCII characters as equivalent. The string comparison will use a case insignificant character comparison. Note that locale dependent case mappings are ignored. This option is therefore not suitable if a properly internationalized or locale aware case mapping should be used. (For example, in a Turkish locale, the upper case equivalent of the lowercase ASCII letter '`i`' is the non-ASCII character '`capital i with dot above`'.) For this case, it is better to apply an uppercase or lowercase conversion on the string before passing it to the `gperf` generated function.

4.3 Options to specify the Language for the Output Code

These options are also available as declarations in the input file (see Section 3.1.1.2 [Gperf Declarations], page 12).

'`-L` *generated-language-name*'
'`--language=`*generated-language-name*'
: Instructs `gperf` to generate code in the language specified by the option's argument. Languages handled are currently:

  '`KR-C`' Old-style K&R C. This language is understood by old-style C compilers and ANSI C compilers, but ANSI C compilers may flag warnings (or even errors) because of lacking '`const`'.

  '`C`' Common C. This language is understood by ANSI C compilers, and also by old-style C compilers, provided that you `#define const` to empty for compilers which don't know about this keyword.

  '`ANSI-C`' ANSI C. This language is understood by ANSI C compilers and C++ compilers.

  '`C++`' C++. This language is understood by C++ compilers.

  The default is C.

'`-a`' This option is supported for compatibility with previous releases of `gperf`. It does not do anything.

'`-g`' This option is supported for compatibility with previous releases of `gperf`. It does not do anything.

4.4 Options for fine tuning Details in the Output Code

Most of these options are also available as declarations in the input file (see Section 3.1.1.2 [Gperf Declarations], page 12).

'`-K` *slot-name*'
'`--slot-name=`*slot-name*'
: This option is only useful when option '`-t`' (or, equivalently, the '`%struct-type`' declaration) has been given. By default, the program assumes the structure component identifier for the keyword is '`name`'. This option allows an arbitrary choice of identifier for this component, although it still must occur as the first field in your supplied `struct`.

'`-F` *initializers*'
'`--initializer-suffix=`*initializers*'
: This option is only useful when option '`-t`' (or, equivalently, the '`%struct-type`' declaration) has been given. It permits to specify initializers for the structure members following *slot-name* in empty hash table entries. The list of initializers should start with a comma. By default, the emitted code will zero-initialize structure members following *slot-name*.

'`-H` *hash-function-name*'
'`--hash-function-name=`*hash-function-name*'
: Allows you to specify the name for the generated hash function. Default name is '`hash`'. This option permits the use of two hash tables in the same file.

'-N *lookup-function-name*'
'--lookup-function-name=*lookup-function-name*'
: Allows you to specify the name for the generated lookup function. Default name is 'in_word_set'. This option permits multiple generated hash functions to be used in the same application.

'-Z *class-name*'
'--class-name=*class-name*'
: This option is only useful when option '-L C++' (or, equivalently, the '%language=C++' declaration) has been given. It allows you to specify the name of generated C++ class. Default name is Perfect_Hash.

'-7'
'--seven-bit'
: This option specifies that all strings that will be passed as arguments to the generated hash function and the generated lookup function will solely consist of 7-bit ASCII characters (bytes in the range 0..127). (Note that the ANSI C functions isalnum and isgraph do *not* guarantee that a byte is in this range. Only an explicit test like 'c >= 'A' && c <= 'Z'' guarantees this.) This was the default in versions of gperf earlier than 2.7; now the default is to support 8-bit and multibyte characters.

'-l'
'--compare-lengths'
: Compare keyword lengths before trying a string comparison. This option is mandatory for binary comparisons (see Section 3.3 [Binary Strings], page 18). It also might cut down on the number of string comparisons made during the lookup, since keywords with different lengths are never compared via strcmp. However, using '-l' might greatly increase the size of the generated C code if the lookup table range is large (which implies that the switch option '-S' or '%switch' is not enabled), since the length table contains as many elements as there are entries in the lookup table.

'-c'
'--compare-strncmp'
: Generates C code that uses the strncmp function to perform string comparisons. The default action is to use strcmp.

'-C'
'--readonly-tables'
: Makes the contents of all generated lookup tables constant, i.e., "readonly". Many compilers can generate more efficient code for this by putting the tables in readonly memory.

'-E'
'--enum'
: Define constant values using an enum local to the lookup function rather than with #defines. This also means that different lookup functions can reside in the same file. Thanks to James Clark <jjc@ai.mit.edu>.

'-I'
'--includes'
: Include the necessary system include file, <string.h>, at the beginning of the code. By default, this is not done; the user must include this header file himself to allow compilation of the code.

'-G'
'--global-table'
: Generate the static table of keywords as a static global variable, rather than hiding it inside of the lookup function (which is the default behavior).

'-P'
'--pic'
: Optimize the generated table for inclusion in shared libraries. This reduces the startup time of programs using a shared library containing the generated code. If the option '-t' (or, equivalently, the '%struct-type' declaration) is also given, the first field of the user-defined struct must be of type 'int', not 'char *', because it will contain offsets into the string pool instead of actual strings. To convert such an offset to a string, you can use the expression 'stringpool + *o*', where *o* is the offset. The string pool name can be changed through the option '--string-pool-name'.

'-Q *string-pool-name*'
'--string-pool-name=*string-pool-name*'
: Allows you to specify the name of the generated string pool created by option '-P'. The default name is 'stringpool'. This option permits the use of two hash tables in the same file, with '-P' and even when the option '-G' (or, equivalently, the '%global-table' declaration) is given.

'--null-strings'
: Use NULL strings instead of empty strings for empty keyword table entries. This reduces the startup time of programs using a shared library containing the generated code (but not as much as option '-P'), at the expense of one more test-and-branch instruction at run time.

'-W *hash-table-array-name*'
'--word-array-name=*hash-table-array-name*'
: Allows you to specify the name for the generated array containing the hash table. Default name is 'wordlist'. This option permits the use of two hash tables in the same file, even when the option '-G' (or, equivalently, the '%global-table' declaration) is given.

'-S *total-switch-statements*'
'--switch=*total-switch-statements*'
: Causes the generated C code to use a switch statement scheme, rather than an array lookup table. This can lead to a reduction in both time and space requirements for some input files. The argument to this option determines how many switch statements are generated. A value of 1 generates 1 switch containing all the elements, a value of 2 generates 2 tables with 1/2 the elements in each switch, etc. This is useful since many C compilers cannot correctly generate code for large switch statements. This option was inspired in part by Keith Bostic's original C program.

'-T'
'--omit-struct-type'
Prevents the transfer of the type declaration to the output file. Use this option if the type is already defined elsewhere.

'-p' This option is supported for compatibility with previous releases of gperf. It does not do anything.

4.5 Options for changing the Algorithms employed by gperf

'-k *selected-byte-positions*'
'--key-positions=*selected-byte-positions*'
Allows selection of the byte positions used in the keywords' hash function. The allowable choices range between 1-255, inclusive. The positions are separated by commas, e.g., '-k 9,4,13,14'; ranges may be used, e.g., '-k 2-7'; and positions may occur in any order. Furthermore, the wildcard '*' causes the generated hash function to consider **all** byte positions in each keyword, whereas '$' instructs the hash function to use the "final byte" of a keyword (this is the only way to use a byte position greater than 255, incidentally).

For instance, the option '-k 1,2,4,6-10,'$'' generates a hash function that considers positions 1,2,4,6,7,8,9,10, plus the last byte in each keyword (which may be at a different position for each keyword, obviously). Keywords with length less than the indicated byte positions work properly, since selected byte positions exceeding the keyword length are simply not referenced in the hash function.

This option is not normally needed since version 2.8 of gperf; the default byte positions are computed depending on the keyword set, through a search that minimizes the number of byte positions.

'-D'
'--duplicates'
Handle keywords whose selected byte sets hash to duplicate values. Duplicate hash values can occur if a set of keywords has the same names, but possesses different attributes, or if the selected byte positions are not well chosen. With the -D option gperf treats all these keywords as part of an equivalence class and generates a perfect hash function with multiple comparisons for duplicate keywords. It is up to you to completely disambiguate the keywords by modifying the generated C code. However, gperf helps you out by organizing the output.

Using this option usually means that the generated hash function is no longer perfect. On the other hand, it permits gperf to work on keyword sets that it otherwise could not handle.

'-m *iterations*'
'--multiple-iterations=*iterations*'
Perform multiple choices of the '-i' and '-j' values, and choose the best results. This increases the running time by a factor of *iterations* but does a good job minimizing the generated table size.

‘`-i` *initial-value*’
‘`--initial-asso=`*initial-value*’
: Provides an initial *value* for the associate values array. Default is 0. Increasing the initial value helps inflate the final table size, possibly leading to more time efficient keyword lookups. Note that this option is not particularly useful when ‘`-S`’ (or, equivalently, ‘`%switch`’) is used. Also, ‘`-i`’ is overridden when the ‘`-r`’ option is used.

‘`-j` *jump-value*’
‘`--jump=`*jump-value*’
: Affects the “jump value”, i.e., how far to advance the associated byte value upon collisions. *Jump-value* is rounded up to an odd number, the default is 5. If the *jump-value* is 0 `gperf` jumps by random amounts.

‘`-n`’
‘`--no-strlen`’
: Instructs the generator not to include the length of a keyword when computing its hash value. This may save a few assembly instructions in the generated lookup table.

‘`-r`’
‘`--random`’
: Utilizes randomness to initialize the associated values table. This frequently generates solutions faster than using deterministic initialization (which starts all associated values at 0). Furthermore, using the randomization option generally increases the size of the table.

‘`-s` *size-multiple*’
‘`--size-multiple=`*size-multiple*’
: Affects the size of the generated hash table. The numeric argument for this option indicates “how many times larger or smaller” the maximum associated value range should be, in relationship to the number of keywords. It can be written as an integer, a floating-point number or a fraction. For example, a value of 3 means “allow the maximum associated value to be about 3 times larger than the number of input keywords”. Conversely, a value of 1/3 means “allow the maximum associated value to be about 3 times smaller than the number of input keywords”. Values smaller than 1 are useful for limiting the overall size of the generated hash table, though the option ‘`-m`’ is better at this purpose.

  If ‘generate switch’ option ‘`-S`’ (or, equivalently, ‘`%switch`’) is *not* enabled, the maximum associated value influences the static array table size, and a larger table should decrease the time required for an unsuccessful search, at the expense of extra table space.

  The default value is 1, thus the default maximum associated value about the same size as the number of keywords (for efficiency, the maximum associated value is always rounded up to a power of 2). The actual table size may vary somewhat, since this technique is essentially a heuristic.

4.6 Informative Output

'-h'
'--help' Prints a short summary on the meaning of each program option. Aborts further program execution.

'-v'
'--version'
Prints out the current version number.

'-d'
'--debug' Enables the debugging option. This produces verbose diagnostics to "standard error" when gperf is executing. It is useful both for maintaining the program and for determining whether a given set of options is actually speeding up the search for a solution. Some useful information is dumped at the end of the program when the '-d' option is enabled.

5 Known Bugs and Limitations with `gperf`

The following are some limitations with the current release of `gperf`:

- The `gperf` utility is tuned to execute quickly, and works quickly for small to medium size data sets (around 1000 keywords). It is extremely useful for maintaining perfect hash functions for compiler keyword sets. Several recent enhancements now enable `gperf` to work efficiently on much larger keyword sets (over 15,000 keywords). When processing large keyword sets it helps greatly to have over 8 megs of RAM.
- The size of the generate static keyword array can get *extremely* large if the input keyword file is large or if the keywords are quite similar. This tends to slow down the compilation of the generated C code, and *greatly* inflates the object code size. If this situation occurs, consider using the '`-S`' option to reduce data size, potentially increasing keyword recognition time a negligible amount. Since many C compilers cannot correctly generate code for large switch statements it is important to qualify the -*S* option with an appropriate numerical argument that controls the number of switch statements generated.
- The maximum number of selected byte positions has an arbitrary limit of 255. This restriction should be removed, and if anyone considers this a problem write me and let me know so I can remove the constraint.

6 Things Still Left to Do

It should be "relatively" easy to replace the current perfect hash function algorithm with a more exhaustive approach; the perfect hash module is essential independent from other program modules. Additional worthwhile improvements include:

- Another useful extension involves modifying the program to generate "minimal" perfect hash functions (under certain circumstances, the current version can be rather extravagant in the generated table size). This is mostly of theoretical interest, since a sparse table often produces faster lookups, and use of the '`-S`' `switch` option can minimize the data size, at the expense of slightly longer lookups (note that the gcc compiler generally produces good code for `switch` statements, reducing the need for more complex schemes).
- In addition to improving the algorithm, it would also be useful to generate an Ada package as the code output, in addition to the current C and C++ routines.

7 Bibliography

[1] Chang, C.C.: *A Scheme for Constructing Ordered Minimal Perfect Hashing Functions* Information Sciences 39(1986), 187-195.

[2] Cichelli, Richard J. *Author's Response to "On Cichelli's Minimal Perfect Hash Functions Method"* Communications of the ACM, 23, 12(December 1980), 729.

[3] Cichelli, Richard J. *Minimal Perfect Hash Functions Made Simple* Communications of the ACM, 23, 1(January 1980), 17-19.

[4] Cook, C. R. and Oldehoeft, R.R. *A Letter Oriented Minimal Perfect Hashing Function* SIGPLAN Notices, 17, 9(September 1982), 18-27.

[5] Cormack, G. V. and Horspool, R. N. S. and Kaiserwerth, M. *Practical Perfect Hashing* Computer Journal, 28, 1(January 1985), 54-58.

[6] Jaeschke, G. *Reciprocal Hashing: A Method for Generating Minimal Perfect Hashing Functions* Communications of the ACM, 24, 12(December 1981), 829-833.

[7] Jaeschke, G. and Osterburg, G. *On Cichelli's Minimal Perfect Hash Functions Method* Communications of the ACM, 23, 12(December 1980), 728-729.

[8] Sager, Thomas J. *A Polynomial Time Generator for Minimal Perfect Hash Functions* Communications of the ACM, 28, 5(December 1985), 523-532

[9] Schmidt, Douglas C. *GPERF: A Perfect Hash Function Generator* Second USENIX C++ Conference Proceedings, April 1990.

[10] Schmidt, Douglas C. *GPERF: A Perfect Hash Function Generator* C++ Report, SIGS 10 10 (November/December 1998).

[11] Sebesta, R.W. and Taylor, M.A. *Minimal Perfect Hash Functions for Reserved Word Lists* SIGPLAN Notices, 20, 12(September 1985), 47-53.

[12] Sprugnoli, R. *Perfect Hashing Functions: A Single Probe Retrieving Method for Static Sets* Communications of the ACM, 20 11(November 1977), 841-850.

[13] Stallman, Richard M. *Using and Porting GNU CC* Free Software Foundation, 1988.

[14] Stroustrup, Bjarne *The C++ Programming Language.* Addison-Wesley, 1986.

[15] Tiemann, Michael D. *User's Guide to GNU C++* Free Software Foundation, 1989.

Concept Index

Table of Contents