User’s Guide to `gperf` 3.1

The GNU Perfect Hash Function Generator
Edition 3.1, 20 December 2009

Douglas C. Schmidt
Bruno Haible

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Preamble

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

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

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

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

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

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

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

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

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

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

TERMS AND CONDITIONS

0. Definitions.

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

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

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

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

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

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

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

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

1. Source Code.

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

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

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

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

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

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

2. Basic Permissions.

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

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

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

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

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

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

4. Conveying Verbatim Copies.

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

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

5. Conveying Modified Source Versions.

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

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

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

6. Conveying Non-Source Forms.

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

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

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

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

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

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

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

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

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

7. Additional Terms.

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

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

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

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

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

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

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

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

8. Termination.

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

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

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

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

9. Acceptance Not Required for Having Copies.

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

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

10. Automatic Licensing of Downstream Recipients.

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

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

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

11. Patents.

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

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

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

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

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

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

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

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

12. No Surrender of Others’ Freedom.

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

13. Use with the GNU Affero General Public License.

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

14. Revised Versions of this License.

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

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

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

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

15. Disclaimer of Warranty.

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

16. Limitation of Liability.

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

17. Interpretation of Sections 15 and 16.

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

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

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

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

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

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

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

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

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

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

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

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

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

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

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 (C89, ISO C90) compilers, ISO C99 compilers, and C++ compilers.

  ‘`C++`’ C++. This language is understood by C++ compilers.

  The default is ANSI-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 22). 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.

‘`%define length-table-name` *name*’
: Allows you to specify the name for the generated array containing the length table. Default name is ‘`lengthtable`’. This option permits the use of two length 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 functions 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 24).

`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.

3.4 The Copyright of the Output

gperf is under GPL, but that does not cause the output produced by gperf to be under GPL. The reason is that the output contains only small pieces of text that come directly from gperf's source code – only about 7 lines long, too small for being significant –, and therefore the output is not a “work based on gperf” (in the sense of the GPL version 3).

On the other hand, the output produced by gperf contains essentially all of the input file. Therefore the output is a “derivative work” of the input (in the sense of U.S. copyright law); and its copyright status depends on the copyright of the input. For most software licenses, the result is that the the output is under the same license, with the same copyright holder, as the input that was passed to gperf.

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 16).

'`-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 16).

'`-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 ANSI-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 16).

'`-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 22). 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.

'`--length-table-name=`*length-table-array-name*'
: Allows you to specify the name for the generated array containing the length table. Default name is '`lengthtable`'. This option permits the use of two length 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