

Contributed by Steven Bosscher ([s.bosscher@gcc.gnu.org](mailto:s.bosscher@gcc.gnu.org)).

# Using GNU Fortran 95

---

Steven Bosscher

---

For the 4.0.0 Version\*

Published by the Free Software Foundation  
59 Temple Place - Suite 330  
Boston, MA 02111-1307, USA

Copyright © 1999-2005 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being “GNU General Public License” and “Funding Free Software”, the Front-Cover texts being (a) (see below), and with the Back-Cover Texts being (b) (see below). A copy of the license is included in the section entitled “GNU Free Documentation License”.

(a) The FSF’s Front-Cover Text is:

A GNU Manual

(b) The FSF’s Back-Cover Text is:

You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.

## Short Contents

Introduction .....	1
GNU GENERAL PUBLIC LICENSE .....	3
GNU Free Documentation License .....	9
Funding Free Software .....	17
1 Getting Started .....	19
2 GFORTRAN and GCC .....	21
3 GFORTRAN and G77 .....	23
4 GNU Fortran 95 Command Options .....	25
5 Project Status .....	33
6 Extensions .....	37
7 Intrinsic Procedures .....	39
8 Contributing .....	57
9 Standards .....	59
Index .....	61



# Table of Contents

<b>Introduction</b>	<b>1</b>
<b>GNU GENERAL PUBLIC LICENSE</b>	<b>3</b>
Preamble	3
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION	4
How to Apply These Terms to Your New Programs	8
<b>GNU Free Documentation License</b>	<b>9</b>
ADDENDUM: How to use this License for your documents	15
<b>Funding Free Software</b>	<b>17</b>
<b>1 Getting Started</b>	<b>19</b>
<b>2 GFORTRAN and GCC</b>	<b>21</b>
<b>3 GFORTRAN and G77</b>	<b>23</b>
<b>4 GNU Fortran 95 Command Options</b>	<b>25</b>
4.1 Option Summary	25
4.2 Options Controlling Fortran Dialect	26
4.3 Options to Request or Suppress Warnings	27
4.4 Options for Debugging Your Program or GNU Fortran	28
4.5 Options for Directory Search	28
4.6 Options for Code Generation Conventions	29
4.7 Environment Variables Affecting GNU Fortran	31
<b>5 Project Status</b>	<b>33</b>
5.1 Compiler Status	33
5.2 Library Status	33
5.3 Proposed Extensions	34
5.3.1 Compiler extensions:	34
5.3.2 Environment Options	35
<b>6 Extensions</b>	<b>37</b>
6.1 Old-style kind specifications	37
6.2 Old-style variable initialization	37

<b>7</b>	<b>Intrinsic Procedures</b>	<b>39</b>
7.1	Introduction to intrinsic procedures	39
7.2	ABORT — Abort the program	39
7.3	ABS — Absolute value	40
7.4	ACHAR — Character in ASCII collating sequence	40
7.5	ACOS — Arccosine function	41
7.6	ADJUSTL — Left adjust a string	41
7.7	ADJUSTR — Right adjust a string	42
7.8	AIMAG — Imaginary part of complex number	42
7.9	AIMT — Imaginary part of complex number	43
7.10	ALL — All values in <i>MASK</i> along <i>DIM</i> are true	43
7.11	ALLOCATED — Status of an allocatable entity	44
7.12	ANINT — Imaginary part of complex number	45
7.13	ANY — Any value in <i>MASK</i> along <i>DIM</i> is true	45
7.14	ASIN — Arcsine function	46
7.15	ATAN — Arctangent function	47
7.16	BESJO — Bessel function of the first kind of order 0	47
7.17	BESJ1 — Bessel function of the first kind of order 1	48
7.18	BESJN — Bessel function of the first kind	48
7.19	BESY0 — Bessel function of the second kind of order 0	49
7.20	BESY1 — Bessel function of the second kind of order 1	49
7.21	BESYN — Bessel function of the second kind	50
7.22	COS — Cosine function	50
7.23	COSH — Hyperbolic cosine function	51
7.24	ERF — Error function	51
7.25	ERFC — Error function	52
7.26	EXP — Exponential function	52
7.27	LOG — Logarithm function	53
7.28	LOG10 — Base 10 logarithm function	53
7.29	SIN — Sine function	54
7.30	SINH — Hyperbolic sine function	54
7.31	SQRT — Square-root function	55
7.32	TAN — Tangent function	55
7.33	TANH — Hyperbolic tangent function	56
<b>8</b>	<b>Contributing</b>	<b>57</b>
8.1	Contributors to GNU Fortran 95	57
8.2	Projects	57
<b>9</b>	<b>Standards</b>	<b>59</b>
	<b>Index</b>	<b>61</b>

# Introduction

This manual documents the use of **gfortran**, the GNU Fortran 95 compiler. You can find in this manual how to invoke **gfortran**, as well as its features and incompatibilities.





# GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.  
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

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

## 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 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 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.

# GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.  
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

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

## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none. The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and



that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

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

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

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

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled ‘‘GNU
Free Documentation License’’.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.



## Funding Free Software

If you want to have more free software a few years from now, it makes sense for you to help encourage people to contribute funds for its development. The most effective approach known is to encourage commercial redistributors to donate.

Users of free software systems can boost the pace of development by encouraging for-a-fee distributors to donate part of their selling price to free software developers—the Free Software Foundation, and others.

The way to convince distributors to do this is to demand it and expect it from them. So when you compare distributors, judge them partly by how much they give to free software development. Show distributors they must compete to be the one who gives the most.

To make this approach work, you must insist on numbers that you can compare, such as, “We will donate ten dollars to the Frobnitz project for each disk sold.” Don’t be satisfied with a vague promise, such as “A portion of the profits are donated,” since it doesn’t give a basis for comparison.

Even a precise fraction “of the profits from this disk” is not very meaningful, since creative accounting and unrelated business decisions can greatly alter what fraction of the sales price counts as profit. If the price you pay is \$50, ten percent of the profit is probably less than a dollar; it might be a few cents, or nothing at all.

Some redistributors do development work themselves. This is useful too; but to keep everyone honest, you need to inquire how much they do, and what kind. Some kinds of development make much more long-term difference than others. For example, maintaining a separate version of a program contributes very little; maintaining the standard version of a program for the whole community contributes much. Easy new ports contribute little, since someone else would surely do them; difficult ports such as adding a new CPU to the GNU Compiler Collection contribute more; major new features or packages contribute the most.

By establishing the idea that supporting further development is “the proper thing to do” when distributing free software for a fee, we can assure a steady flow of resources into making more free software.

Copyright © 1994 Free Software Foundation, Inc.

Verbatim copying and redistribution of this section is permitted without royalty; alteration is not permitted.





# 1 Getting Started

Gfortran is the GNU Fortran 95 compiler front end, designed initially as a free replacement for, or alternative to, the unix `f95` command; `gfortran` is command you'll use to invoke the compiler.

*Gfortran is not yet a fully conformant Fortran 95 compiler.* It can generate code for most constructs and expressions, but work remains to be done. In particular, there are known deficiencies with `ENTRY`, `NAMelist`, and sophisticated use of `MODULES`, `POINTERS` and `DERIVED TYPES`. For those whose Fortran codes conform to either the Fortran 77 standard or the GNU Fortran 77 language, we recommend to use `g77` from GCC 3.4. We recommend that distributors continue to provide packages of `g77-3.4` until we announce that `gfortran` fully replaces `g77`. The `gfortran` developers welcome any feedback on user experience with `gfortran` at `fortran@gcc.gnu.org`.

When `gfortran` is finished, it will do everything you expect from any decent compiler:

- Read a user's program, stored in a file and containing instructions written in Fortran 77, Fortran 90 or Fortran 95. This file contains *source code*.
- Translate the user's program into instructions a computer can carry out more quickly than it takes to translate the instructions in the first place. The result after compilation of a program is *machine code*, code designed to be efficiently translated and processed by a machine such as your computer. Humans usually aren't as good writing machine code as they are at writing Fortran (or C++, Ada, or Java), because is easy to make tiny mistakes writing machine code.
- Provide the user with information about the reasons why the compiler is unable to create a binary from the source code. Usually this will be the case if the source code is flawed. When writing Fortran, it is easy to make big mistakes. The Fortran 90 requires that the compiler can point out mistakes to the user. An incorrect usage of the language causes an *error message*.

The compiler will also attempt to diagnose cases where the user's program contains a correct usage of the language, but instructs the computer to do something questionable. This kind of diagnostics message is called a *warning message*.

- Provide optional information about the translation passes from the source code to machine code. This can help a user of the compiler to find the cause of certain bugs which may not be obvious in the source code, but may be more easily found at a lower level compiler output. It also helps developers to find bugs in the compiler itself.
- Provide information in the generated machine code that can make it easier to find bugs in the program (using a debugging tool, called a *debugger*, such as the GNU Debugger `gdb`).
- Locate and gather machine code already generated to perform actions requested by statements in the user's program. This machine code is organized into *modules* and is located and *linked* to the user program.

Gfortran consists of several components:

- A version of the `gcc` command (which also might be installed as the system's `cc` command) that also understands and accepts Fortran source code. The `gcc` command is the *driver* program for all the languages in the GNU Compiler Collection (GCC); With

`gcc`, you can compile the source code of any language for which a front end is available in GCC.

- The `gfortran` command itself, which also might be installed as the system's `f95` command. `gfortran` is just another driver program, but specifically for the Fortran 95 compiler only. The difference with `gcc` is that `gfortran` will automatically link the correct libraries to your program.
- A collection of run-time libraries. These libraries contain the machine code needed to support capabilities of the Fortran language that are not directly provided by the machine code generated by the `gfortran` compilation phase, such as intrinsic functions and subroutines, and routines for interaction with files and the operating system.
- The Fortran compiler itself, (`f951`). This is the `gfortran` parser and code generator, linked to and interfaced with the GCC backend library. `f951` “translates” the source code to assembler code. You would typically not use this program directly; instead, the `gcc` or `gfortran` driver programs will call it for you.

## 2 GFORTRAN and GCC

GCC used to be the GNU “C” Compiler, but is now known as the *GNU Compiler Collection*. GCC provides the GNU system with a very versatile compiler middle end (shared optimization passes), and with back ends (code generators) for many different computer architectures and operating systems. The code of the middle end and back end are shared by all compiler front ends that are in the GNU Compiler Collection.

A GCC front end is essentially a source code parser and a pass to generate a representation of the semantics of the program in the source code in the GCC language independent intermediate language, called *GENERIC*.

The parser takes a source file written in a particular computer language, reads and parses it, and tries to make sure that the source code conforms to the language rules. Once the correctness of a program has been established, the compiler will build a data structure known as the *Abstract Syntax tree*, or just *AST* or “tree” for short. This data structure represents the whole program or a subroutine or a function. The “tree” is passed to the GCC middle end, which will perform optimization passes on it, pass the optimized AST and generate assembly for the program unit.

Different phases in this translation process can be, and in fact *are* merged in many compiler front ends. GNU Fortran 95 has a strict separation between the parser and code generator.

The goal of the gfortran project is to build a new front end for GCC: A Fortran 95 front end. In a non-gfortran installation, gcc will not be able to compile Fortran 95 source code (only the “C” front end has to be compiled if you want to build GCC, all other languages are optional). If you build GCC with gfortran, gcc will recognize ‘.f/.f90/.f95’ source files and accepts Fortran 95 specific command line options.



### 3 GFORTRAN and G77

Why do we write a compiler front end from scratch? There's a fine Fortran 77 compiler in the GNU Compiler Collection that accepts some features of the Fortran 90 standard as extensions. Why not start from there and revamp it?

One of the reasons is that Craig Burley, the author of G77, has decided to stop working on the G77 front end. On Craig explains the reasons for his decision to stop working on G77 (<http://world.std.com/~burley/g77-why.html>) in one of the pages in his homepage. Among the reasons is a lack of interest in improvements to `g77`. Users appear to be quite satisfied with `g77` as it is. While `g77` is still being maintained (by Toon Moene), it is unlikely that sufficient people will be willing to completely rewrite the existing code.

But there are other reasons to start from scratch. Many people, including Craig Burley, no longer agreed with certain design decisions in the G77 front end. Also, the interface of `g77` to the back end is written in a style which is confusing and not up to date on recommended practice. In fact, a full rewrite had already been planned for GCC 3.0.

When Craig decided to stop, it just seemed to be a better idea to start a new project from scratch, because it was expected to be easier to maintain code we develop ourselves than to do a major overhaul of `g77` first, and then build a Fortran 95 compiler out of it.



## 4 GNU Fortran 95 Command Options

The `gfortran` command supports all the options supported by the `gcc` command. Only options specific to `gfortran` are documented here.

*Gfortran is not yet a fully conformant Fortran 95 compiler.* It can generate code for most constructs and expressions, but work remains to be done. In particular, there are known deficiencies with `ENTRY`, `NAMelist`, and sophisticated use of `MODULES`, `POINTERS` and `DERIVED TYPES`. For those whose Fortran codes conform to either the Fortran 77 standard or the GNU Fortran 77 language, we recommend to use `g77` from GCC 3.4. We recommend that distributors continue to provide packages of `g77-3.4` until we announce that `gfortran` fully replaces `g77`. The `gfortran` developers welcome any feedback on user experience with `gfortran` at [fortran@gcc.gnu.org](mailto:fortran@gcc.gnu.org).

See section “GCC Command Options” in *Using the GNU Compiler Collection (GCC)*, for information on the non-Fortran-specific aspects of the `gcc` command (and, therefore, the `gfortran` command).

All `gcc` and `gfortran` options are accepted both by `gfortran` and by `gcc` (as well as any other drivers built at the same time, such as `g++`), since adding `gfortran` to the `gcc` distribution enables acceptance of `gfortran` options by all of the relevant drivers.

In some cases, options have positive and negative forms; the negative form of ‘`-ffoo`’ would be ‘`-fno-foo`’. This manual documents only one of these two forms, whichever one is not the default.

### 4.1 Option Summary

Here is a summary of all the options specific to GNU Fortran, grouped by type. Explanations are in the following sections.

#### *Fortran Language Options*

See Section 4.2 [Options Controlling Fortran Dialect], page 26.

```
-ffree-form -fno-fixed-form
-fdollar-ok -fimplicit-none -fmax-identifier-length
-std=std -ffixed-line-length-n -ffixed-line-length-none
-fdefault-double-8 -fdefault-integer-8 -fdefault-real-8
```

#### *Warning Options*

See Section 4.3 [Options to Request or Suppress Warnings], page 27.

```
-fsyntax-only -pedantic -pedantic-errors
-w -Wall -Waliasing -Wconversion
-Wimplicit-interface -Wnonstd-intrinsics -Wsurprising -Wunderflow
-Wunused-labels -Wline-truncation
-Werror -W
```

#### *Debugging Options*

See Section 4.4 [Options for Debugging Your Program or GCC], page 28.

```
-fdump-parse-tree
```

#### *Directory Options*

See Section 4.5 [Options for Directory Search], page 28.

```
-Idir -Mdir
```

#### *Code Generation Options*

See Section 4.6 [Options for Code Generation Conventions], page 29.

```
-fno-underscoring -fno-second-underscore
-fbounds-check -fmax-stack-var-size=n
-fpackderived -frepack-arrays
```

## 4.2 Options Controlling Fortran Dialect

The following options control the dialect of Fortran that the compiler accepts:

**-ffree-form**

**-ffixed-form**

Specify the layout used by the the source file. The free form layout was introduced in Fortran 90. Fixed form was traditionally used in older Fortran programs.

**-fdefault-double-8**

Set the "DOUBLE PRECISION" type to an 8 byte wide.

**-fdefault-integer-8**

Set the default integer and logical types to an 8 byte wide type. Do nothing if this is already the default.

**-fdefault-real-8**

Set the default real type to an 8 byte wide type. Do nothing if this is already the default.

**-fdollar-ok**

Allow '\$' as a valid character in a symbol name.

**-ffixed-line-length-n**

Set column after which characters are ignored in typical fixed-form lines in the source file, and through which spaces are assumed (as if padded to that length) after the ends of short fixed-form lines.

Popular values for *n* include 72 (the standard and the default), 80 (card image), and 132 (corresponds to "extended-source" options in some popular compilers). *n* may be 'none', meaning that the entire line is meaningful and that continued character constants never have implicit spaces appended to them to fill out the line. '-ffixed-line-length-0' means the same thing as '-ffixed-line-length-none'.

**-fmax-identifier-length=n**

Specify the maximum allowed identifier length. Typical values are 31 (Fortran 95) and 63 (Fortran 200x).

**-fimplicit-none**

Specify that no implicit typing is allowed, unless overridden by explicit 'IMPLICIT' statements. This is the equivalent of adding 'implicit none' to the start of every procedure.

**-std=std** Conform to the specified standard. Allowed values for *std* are 'gnu' and 'f95'.



### 4.3 Options to Request or Suppress Warnings

Warnings are diagnostic messages that report constructions which are not inherently erroneous but which are risky or suggest there might have been an error.

You can request many specific warnings with options beginning ‘-W’, for example ‘-Wimplicit’ to request warnings on implicit declarations. Each of these specific warning options also has a negative form beginning ‘-Wno-’ to turn off warnings; for example, ‘-Wno-implicit’. This manual lists only one of the two forms, whichever is not the default.

These options control the amount and kinds of warnings produced by GNU Fortran:

**-fsyntax-only**

Check the code for syntax errors, but don’t do anything beyond that.

**-pedantic**

Issue warnings for uses of extensions to FORTRAN 95. ‘-pedantic’ also applies to C-language constructs where they occur in GNU Fortran source files, such as use of ‘\e’ in a character constant within a directive like ‘#include’.

Valid FORTRAN 95 programs should compile properly with or without this option. However, without this option, certain GNU extensions and traditional Fortran features are supported as well. With this option, many of them are rejected.

Some users try to use ‘-pedantic’ to check programs for conformance. They soon find that it does not do quite what they want—it finds some nonstandard practices, but not all. However, improvements to `gfortran` in this area are welcome.

This should be used in conjunction with `-std=std`.

**-pedantic-errors**

Like ‘-pedantic’, except that errors are produced rather than warnings.

**-w**

Inhibit all warning messages.

**-Wall**

Enables commonly used warning options that which pertain to usage that we recommend avoiding and that we believe is easy to avoid. This currently includes ‘-Wunused-labels’, ‘-Waliasing’, ‘-Wsurprising’, ‘-Wnonstd-intrinsic’ and ‘-Wline-truncation’.

**-Waliasing**

Warn about possible aliasing of dummy arguments. The following example will trigger the warning as it would be illegal to `bar` to modify either parameter.

```
INTEGER A
CALL BAR(A,A)
```

**-Wconversion**

Warn about implicit conversions between different types.

**-Wimplicit-interface**

Warn about when procedure are called without an explicit interface. Note this only checks that an explicit interface is present. It does not check that the declared interfaces are consistent across program units.

**-Wnonstd-intrinsic**

Warn if the user tries to use an intrinsic that does not belong to the standard the user has chosen via the `-std` option.

**-Wsurprising**

Produce a warning when “suspicious” code constructs are encountered. While technically legal these usually indicate that an error has been made.

This currently produces a warning under the following circumstances:

- An `INTEGER SELECT` construct has a `CASE` that can never be matched as its lower value is greater than its upper value.
- A `LOGICAL SELECT` construct has three `CASE` statements.

**-Wunderflow**

Produce a warning when numerical constant expressions are encountered, which yield an `UNDERFLOW` during compilation.

**-Wunused-labels**

Warn whenever a label is defined but never referenced.

**-Werror** Turns all warnings into errors.**-W** Turns on “extra warnings” and, if optimization is specified via `‘-O’`, the `‘-Wuninitialized’` option. (This might change in future versions of `gfortran`

See section “Options to Request or Suppress Warnings” in *Using the GNU Compiler Collection (GCC)*, for information on more options offered by the GBE shared by `gfortran`, `gcc` and other GNU compilers.

Some of these have no effect when compiling programs written in Fortran.

## 4.4 Options for Debugging Your Program or GNU Fortran

GNU Fortran has various special options that are used for debugging either your program or `gfortran`

**-fdump-parse-tree**

Output the internal parse tree before starting code generation. Only really useful for debugging `gfortran` itself.

See section “Options for Debugging Your Program or GCC” in *Using the GNU Compiler Collection (GCC)*, for more information on debugging options.

## 4.5 Options for Directory Search

These options affect how `gfortran` searches for files specified via the `INCLUDE` directive, and where it searches for previously compiled modules.

It also affects the search paths used by `cpp` when used to preprocess Fortran source.

**-Idir** These affect interpretation of the `INCLUDE` directive (as well as of the `#include` directive of the `cpp` preprocessor).

Also note that the general behavior of `‘-I’` and `INCLUDE` is pretty much the same as of `‘-I’` with `#include` in the `cpp` preprocessor, with regard to looking for `‘header.gcc’` files and other such things.

This path is also used to search for ‘.mod’ files when previously compiled modules are required by a USE statement.

See section “Options for Directory Search” in *Using the GNU Compiler Collection (GCC)*, for information on the ‘-I’ option.

**-Mdir**

**-Jdir** This option specifies where to put ‘.mod’ files for compiled modules. It is also added to the list of directories to searched by an USE statement.

The default is the current directory.

‘-J’ is an alias for ‘-M’ to avoid conflicts with existing GCC options.

## 4.6 Options for Code Generation Conventions

These machine-independent options control the interface conventions used in code generation.

Most of them have both positive and negative forms; the negative form of ‘-ffoo’ would be ‘-fno-foo’. In the table below, only one of the forms is listed—the one which is not the default. You can figure out the other form by either removing ‘no-’ or adding it.

**-fno-underscoring**

Do not transform names of entities specified in the Fortran source file by appending underscores to them.

With ‘-funderscoring’ in effect, **gfortran** appends two underscores to names with underscores and one underscore to external names with no underscores. (**gfortran** also appends two underscores to internal names with underscores to avoid naming collisions with external names. The ‘-fno-second-underscore’ option disables appending of the second underscore in all cases.)

This is done to ensure compatibility with code produced by many UNIX Fortran compilers, including **f2c** which perform the same transformations.

Use of ‘-fno-underscoring’ is not recommended unless you are experimenting with issues such as integration of (GNU) Fortran into existing system environments (vis-a-vis existing libraries, tools, and so on).

For example, with ‘-funderscoring’, and assuming other defaults like ‘-fcase-lower’ and that ‘j()’ and ‘max\_count()’ are external functions while ‘my\_var’ and ‘lvar’ are local variables, a statement like

```
I = J() + MAX_COUNT (MY_VAR, LVAR)
```

is implemented as something akin to:

```
i = j_() + max_count_(&my_var_, &lvar);
```

With ‘-fno-underscoring’, the same statement is implemented as:

```
i = j() + max_count(&my_var, &lvar);
```

Use of ‘-fno-underscoring’ allows direct specification of user-defined names while debugging and when interfacing **gfortran** code with other languages.

Note that just because the names match does *not* mean that the interface implemented by **gfortran** for an external name matches the interface implemented by some other language for that same name. That is, getting code produced by

**gfortran** to link to code produced by some other compiler using this or any other method can be only a small part of the overall solution—getting the code generated by both compilers to agree on issues other than naming can require significant effort, and, unlike naming disagreements, linkers normally cannot detect disagreements in these other areas.

Also, note that with ‘**-fno-underscoring**’, the lack of appended underscores introduces the very real possibility that a user-defined external name will conflict with a name in a system library, which could make finding unresolved-reference bugs quite difficult in some cases—they might occur at program run time, and show up only as buggy behavior at run time.

In future versions of **gfortran** we hope to improve naming and linking issues so that debugging always involves using the names as they appear in the source, even if the names as seen by the linker are mangled to prevent accidental linking between procedures with incompatible interfaces.

#### **-fno-second-underscore**

Do not append a second underscore to names of entities specified in the Fortran source file.

This option has no effect if ‘**-fno-underscoring**’ is in effect.

Otherwise, with this option, an external name such as ‘**MAX\_COUNT**’ is implemented as a reference to the link-time external symbol ‘**max\_count\_**’, instead of ‘**max\_count\_\_**’.

#### **-fbounds-check**

Enable generation of run-time checks for array subscripts and against the declared minimum and maximum values. It also checks array indices for assumed and deferred shape arrays against the actual allocated bounds.

In the future this may also include other forms of checking, eg. checking substring references.

#### **-fmax-stack-var-size=n**

This option specifies the size in bytes of the largest array that will be put on the stack.

This option currently only affects local arrays declared with constant bounds, and may not apply to all character variables. Future versions of **gfortran** may improve this behavior.

The default value for *n* is 32768.

#### **-fpackderived**

This option tells **gfortran** to pack derived type members as closely as possible. Code compiled with this option is likely to be incompatible with code compiled without this option, and may execute slower.

#### **-frepack-arrays**

In some circumstances **gfortran** may pass assumed shape array sections via a descriptor describing a discontinuous area of memory. This option adds code to the function prologue to repack the data into a contiguous block at runtime.

This should result in faster accesses to the array. However it can introduce significant overhead to the function call, especially when the passed data is discontinuous.

See section “Options for Code Generation Conventions” in *Using the GNU Compiler Collection (GCC)*, for information on more options offered by the GBE shared by `gfortran`, `gcc` and other GNU compilers.

## 4.7 Environment Variables Affecting GNU Fortran

GNU Fortran 95 currently does not make use of any environment variables to control its operation above and beyond those that affect the operation of `gcc`.

See section “Environment Variables Affecting GCC” in *Using the GNU Compiler Collection (GCC)*, for information on environment variables.



## 5 Project Status

As soon as gfortran can parse all of the statements correctly, it will be in the “larva” state. When we generate code, the “puppa” state. When gfortran is done, we’ll see if it will be a beautiful butterfly, or just a big bug....

–Andy Vaught, April 2000

The start of the GNU Fortran 95 project was announced on the GCC homepage in March 18, 2000 (even though Andy had already been working on it for a while, or course).

Gfortran is currently reaching the stage where it is able to compile real world programs. However it is still under development and has many rough edges.

### 5.1 Compiler Status

*Front end* This is the part of gfortran which parses a source file, verifies that it is valid Fortran 95, performs compile time replacement of constants (PARAMETER variables) and reads and generate module files. This is almost complete. Every Fortran 95 source should be accepted, and most non-Fortran 95 source should be rejected. If you find a source file where this is not true, please tell us. You can use the `-fsyntax-only` switch to make gfortran quit after running the front end, effectively reducing it to a syntax checker.

*Middle end interface*

These are the parts of gfortran that take the parse tree generated by the front end and translate it to the GENERIC form required by the GCC back end. Work is ongoing in these parts of gfortran, but a large part has already been completed.

### 5.2 Library Status

Some intrinsic functions map directly to library functions, and in most cases the name of the library function used depends on the type of the arguments. For some intrinsics we generate inline code, and for others, such as `sin`, `cos` and `sqrt`, we rely on the backend to use special instructions in the floating point unit of the CPU if available, or to fall back to a call to `libm` if these are not available.

Implementation of some non-elemental intrinsic functions (eg. `DOT_PRODUCT`, `AVERAGE`) is not yet optimal. This is hard because we have to make decisions whether to use inline code (good for small arrays as no function call overhead occurs) or generate function calls (good for large arrays as it allows use of hand-optimized assembly routines, SIMD instructions, etc.)

The IO library is still under development. The following features should be usable for real programs:

- List directed
- Unformatted sequential

Usable with bugs:

- Formatted sequential (‘T’ edit descriptor, and others)
- Namelist (can read a namelist that it writes, but not free-form)

Not recommended:

- Unformatted direct access
- Formatted direct access

Many Fortran programs only use a small subset of the available IO capabilities, so your mileage may vary.

## 5.3 Proposed Extensions

Here’s a list of proposed extensions for `gfortran`, in no particular order. Most of these are necessary to be fully compatible with existing Fortran compilers, but they are not part of the official J3 Fortran 95 standard.

### 5.3.1 Compiler extensions:

- Flag for defining the kind number for default logicals.
- User-specified alignment rules for structures.
- Flag to generate a `Makefile` info.
- Automatically extend single precision constants to double.
- Cray pointers (this was high on the `g77` wishlist).
- Compile code that conserves memory by dynamically allocating common and module storage either on stack or heap.
- Flag to cause the compiler to distinguish between upper and lower case names. The Fortran 95 standard does not distinguish them.
- Compile switch for changing the interpretation of a backslash from a character to “C”-style escape characters.
- Compile flag to generate code for array conformance checking (suggest `-CC`).
- User control of symbol names (underscores, etc).
- Compile setting for maximum size of stack frame size before spilling parts to static or heap.
- Flag to force local variables into static space.
- Flag to force local variables onto stack.
- Flag to compile lines beginning with “D”.
- Flag to ignore lines beginning with “D”.
- Flag for maximum errors before ending compile.
- Generate code to check for null pointer dereferences – prints locus of dereference instead of segfaulting. There was some discussion about this option in the `g95` development mailing list.
- Allow setting default unit number.
- Option to initialize of otherwise uninitialized integer and floating point variables.
- Support for OpenMP directives. This also requires support from the runtime library and the rest of the compiler.
- Support for Fortran 200x. This includes several new features including floating point exceptions, extended use of allocatable arrays, C interoperability, Parameterizer data types and function pointers.



### 5.3.2 Environment Options

- Pluggable library modules for random numbers, linear algebra. LA should use BLAS calling conventions.
- Environment variables controlling actions on arithmetic exceptions like overflow, underflow, precision loss – Generate NaN, abort, default. action.
- Set precision for fp units that support it (i387).
- Variables for setting fp rounding mode.
- Support old style namelists ending in \$end or &end.
- Variable to fill uninitialized variables with a user-defined bit pattern.
- Environment variable controlling filename that is opened for that unit number.
- Environment variable to clear/trash memory being freed.
- Environment variable to control tracing of allocations and frees.
- Environment variable to display allocated memory at normal program end.
- Environment variable for filename for \* IO-unit.
- Environment variable for temporary file directory.
- Environment variable forcing standard output to be line buffered (unix).
- Variable for swapping endianness during unformatted read.
- Variable for swapping Endianness during unformatted write.



## 6 Extensions

`gfortran` implements a number of extensions over standard Fortran. This chapter contains information on their syntax and meaning.

### 6.1 Old-style kind specifications

`gfortran` allows old-style kind specifications in declarations. These look like:

```
TYPESPEC*k x,y,z
```

where `TYPESPEC` is a basic type, and where `k` is a valid kind number for that type. The statement then declares `x`, `y` and `z` to be of type `TYPESPEC` with kind `k`. In other words, it is equivalent to the standard conforming declaration

```
TYPESPEC(k) x,y,z
```

### 6.2 Old-style variable initialization

`gfortran` allows old-style initialization of variables of the form:

```
INTEGER*4 i/1/,j/2/  
REAL*8 x(2,2) /3*0.,1./
```

These are only allowed in declarations without double colons (`::`), as these were introduced in Fortran 90 which also introduced a new syntax for variable initializations. The syntax for the individual initializers is as for the `DATA` statement, but unlike in a `DATA` statement, an initializer only applies to the variable immediately preceding. In other words, something like `INTEGER I,J/2,3/` is not valid.

Examples of standard conforming code equivalent to the above example, are:

```
! Fortran 90  
  INTEGER(4) :: i = 1, j = 2  
  REAL(8) :: x(2,2) = RESHAPE((/0.,0.,0.,1./),SHAPE(x))  
! Fortran 77  
  INTEGER i, j  
  DOUBLE PRECISION x(2,2)  
  DATA i,j,x /1,2,3*0.,1./
```



## 7 Intrinsic Procedures

This portion of the document is incomplete and undergoing massive expansion and editing. All contributions and corrections are strongly encouraged.

### 7.1 Introduction to intrinsic procedures

Gfortran provides a rich set of intrinsic procedures that includes all the intrinsic procedures required by the Fortran 95 standard, a set of intrinsic procedures for backwards compatibility with Gnu Fortran 77 (i.e., `g77`), and a small selection of intrinsic procedures from the Fortran 2003 standard. Any description here, which conflicts with a description in either the Fortran 95 standard or the Fortran 2003 standard, is unintentional and the standard(s) should be considered authoritative.

The enumeration of the `KIND` type parameter is processor defined in the Fortran 95 standard. Gfortran defines the default integer type and default real type by `INTEGER(KIND=4)` and `REAL(KIND=4)`, respectively. The standard mandates that both data types shall have another kind, which have more precision. On typical target architectures supported by `gfortran`, this kind type parameter is `KIND=8`. Hence, `REAL(KIND=8)` and `DOUBLE PRECISION` are equivalent. In the description of generic intrinsic procedures, the kind type parameter will be specified by `KIND=*`, and in the description of specific names for an intrinsic procedure the kind type parameter will be explicitly given (e.g., `REAL(KIND=4)` or `REAL(KIND=8)`). Finally, for brevity the optional `KIND=` syntax will be omitted.

Many of the intrinsic procedures take one or more optional arguments. This document follows the convention used in the Fortran 95 standard, and denotes such arguments by square brackets.

Gfortran offers the `'-std=f95'` and `'-std=gnu'` options, which can be used to restrict the set of intrinsic procedures to a given standard. By default, `gfortran` sets the `'-std=gnu'` option, and so all intrinsic procedures described here are accepted. There is one caveat. For a select group of intrinsic procedures, `g77` implemented both a function and a subroutine. Both classes have been implemented in `gfortran` for backwards compatibility with `g77`. It is noted here that these functions and subroutines cannot be intermixed in a given subprogram. In the descriptions that follow, the applicable option(s) is noted.

### 7.2 ABORT — Abort the program

*Description:*

ABORT causes immediate termination of the program. On operating systems that support a core dump, ABORT will produce a core dump, which is suitable for debugging purposes.

*Option:*     gnu

*Type:*       non-elemental subroutine

*Syntax:*     CALL ABORT

*Return value:*

Does not return.

*Example:*

```

program test_abort
  integer :: i = 1, j = 2
  if (i /= j) call abort
end program test_abort

```

### 7.3 ABS — Absolute value

*Description:*

ABS(X) computes the absolute value of X.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* X = ABS(X)

*Arguments:*

X                      The type of the argument shall be an INTEGER(\*), REAL(\*), or COMPLEX(\*) .

*Return value:*

The return value is of the same type and kind as the argument except the return value is REAL(\*) for a COMPLEX(\*) argument.

*Example:*

```

program test_abs
  integer :: i = -1
  real :: x = -1.e0
  complex :: z = (-1.e0,0.e0)
  i = abs(i)
  x = abs(x)
  z = abs(z)
end program test_abs

```

*Specific names:*

Name	Argument	Return type	Option
CABS(Z)	COMPLEX(4) Z	REAL(4)	f95, gnu
DABS(X)	REAL(8) X	REAL(8)	f95, gnu
IABS(I)	INTEGER(4) I	INTEGER(4)	f95, gnu
ZABS(Z)	COMPLEX(8) Z	COMPLEX(8)	gnu
CDABS(Z)	COMPLEX(8) Z	COMPLEX(8)	gnu

### 7.4 ACHAR — Character in ASCII collating sequence

*Description:*

ACHAR(I) returns the character located at position I in the ASCII collating sequence.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* C = ACHAR(I)

*Arguments:*

*I*                      The type shall be an `INTEGER(*)`.

*Return value:*

The return value is of type `CHARACTER` with a length of one. The kind type parameter is the same as `KIND('A')`.

*Example:*

```
program test_achar
  character c
  c = achar(32)
end program test_achar
```

**7.5 ACOS — Arccosine function***Description:*

`ACOS(X)` computes the arccosine of its *X*.

*Option:*      f95, gnu

*Type:*        elemental function

*Syntax:*      `X = ACOS(X)`

*Arguments:*

*X*                      The type shall be an `REAL(*)`, and a magnitude that is less than one.

*Return value:*

The return value is of type `REAL(*)` and it lies in the range  $0 \leq \arccos(x) \leq \pi$ .  
The kind type parameter is the same as *X*.

*Example:*

```
program test_acos
  real(8) :: x = 0.866_8
  x = achar(x)
end program test_acos
```

*Specific names:*

Name	Argument	Return type	Option
<code>DACOS(X)</code>	<code>REAL(8) X</code>	<code>REAL(8)</code>	f95, gnu

**7.6 ADJUSTL — Left adjust a string***Description:*

`ADJUSTL(STR)` will left adjust a string by removing leading spaces. Spaces are inserted at the end of the string as needed.

*Option:*      f95, gnu

*Type:*        elemental function

*Syntax:*      `STR = ADJUSTL(STR)`

*Arguments:*

*STR*                      The type shall be `CHARACTER`.

*Return value:*

The return value is of type `CHARACTER` where leading spaces are removed and the same number of spaces are inserted on the end of *STR*.

*Example:*

```
program test_adjustl
  character(len=20) :: str = '  gfortran'
  str = adjustl(str)
  print *, str
end program test_adjustl
```

## 7.7 ADJUSTR — Right adjust a string

*Description:*

`ADJUSTR(STR)` will right adjust a string by removing trailing spaces. Spaces are inserted at the start of the string as needed.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* `STR = ADJUSTR(STR)`

*Arguments:*

*STR*                      The type shall be `CHARACTER`.

*Return value:*

The return value is of type `CHARACTER` where trailing spaces are removed and the same number of spaces are inserted at the start of *STR*.

*Example:*

```
program test_adjustr
  character(len=20) :: str = 'gfortran'
  str = adjustr(str)
  print *, str
end program test_adjustr
```

## 7.8 AIMAG — Imaginary part of complex number

*Description:*

`AIMAG(Z)` yields the imaginary part of complex argument *Z*.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* `X = AIMAG(Z)`

*Arguments:*

*Z*                              The type of the argument shall be `COMPLEX(*)`.

*Return value:*

The return value is of type real with the kind type parameter of the argument.

*Example:*



```

program test_aimag
  complex(4) z4
  complex(8) z8
  z4 = cmplx(1.e0_4, 0.e0_4)
  z8 = cmplx(0.e0_8, 1.e0_8)
  print *, aimag(z4), dimag(z8)
end program test_aimag

```

*Specific names:*

Name	Argument	Return type	Option
DIMAG(Z)	COMPLEX(8) Z	REAL(8)	f95, gnu

## 7.9 AINT — Imaginary part of complex number

*Description:*

AINT(X [, KIND]) truncates its argument to a whole number.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* X = AINT(X)  
X = AINT(X, KIND)

*Arguments:*

X                      The type of the argument shall be REAL(\*).  
KIND                    (Optional) KIND shall be a scalar integer initialization expression.

*Return value:*

The return value is of type real with the kind type parameter of the argument if the optional *KIND* is absence; otherwise, the kind type parameter will be given by *KIND*. If the magnitude of *X* is less than one, then AINT(*X*) returns zero. If the magnitude is equal to or greater than one, then it returns the largest whole number that does not exceed its magnitude. The sign is the same as the sign of *X*.

*Example:*

```

program test_aint
  real(4) x4
  real(8) x8
  x4 = 1.234E0_4
  x8 = 4.321_8
  print *, aint(x4), dint(x8)
  x8 = aint(x4,8)
end program test_aint

```

*Specific names:*

Name	Argument	Return type	Option
DINT(X)	REAL(8) X	REAL(8)	f95, gnu

## 7.10 ALL — All values in MASK along DIM are true

*Description:*

ALL(MASK [, DIM]) determines if all the values are true in MASK in the array along dimension DIM.

*Option:* f95, gnu

*Type:* transformational function

*Syntax:* L = ALL(MASK)  
L = ALL(MASK, DIM)

*Arguments:*

*MASK* The type of the argument shall be LOGICAL(\*) and it shall not be scalar.

*DIM* (Optional) *DIM* shall be a scalar integer with a value that lies between one and the rank of *MASK*.

*Return value:*

ALL(MASK) returns a scalar value of type LOGICAL(\*) where the kind type parameter is the same as the kind type parameter of *MASK*. If *DIM* is present, then ALL(MASK, DIM) returns an array with the rank of *MASK* minus 1. The shape is determined from the shape of *MASK* where the *DIM* dimension is elided.

- (A) ALL(MASK) is true if all elements of *MASK* are true. It also is true if *MASK* has zero size; otherwise, it is false.
- (B) If the rank of *MASK* is one, then ALL(MASK, DIM) is equivalent to ALL(MASK). If the rank is greater than one, then ALL(MASK, DIM) is determined by applying ALL to the array sections.

*Example:*

```

program test_all
  logical l
  l = all((/.true., .true., .true./))
  print *, l
  call section
contains
  subroutine section
    integer a(2,3), b(2,3)
    a = 1
    b = 1
    b(2,2) = 2
    print *, all(a .eq. b, 1)
    print *, all(a .eq. b, 2)
  end subroutine section
end program test_all

```

## 7.11 ALLOCATED — Status of an allocatable entity

*Description:*

ALLOCATED(X) checks the status of whether *X* is allocated.

*Option:* f95, gnu

*Type:* inquiry function

*Syntax:* L = ALLOCATED(X)

*Arguments:*

*X* The argument shall be an ALLOCATABLE array.

*Return value:*

The return value is a scalar LOGICAL with the default logical kind type parameter. If *X* is allocated, ALLOCATED(*X*) is .TRUE.; otherwise, it returns the .TRUE.

*Example:*

```
program test_allocated
  integer :: i = 4
  real(4), allocatable :: x(:)
  if (allocated(x) .eqv. .false.) allocate(x(i))
end program test_allocated
```

## 7.12 ANINT — Imaginary part of complex number

*Description:*

ANINT(*X* [, *KIND*]) rounds its argument to the nearest whole number.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* *X* = ANINT(*X*)  
*X* = ANINT(*X*, *KIND*)

*Arguments:*

*X*                      The type of the argument shall be REAL(\*).  
*KIND*                    (Optional) *KIND* shall be a scalar integer initialization expression.

*Return value:*

The return value is of type real with the kind type parameter of the argument if the optional *KIND* is absence; otherwise, the kind type parameter will be given by *KIND*. If *X* is greater than zero, then ANINT(*X*) returns AINT(*X*+0.5). If *X* is less than or equal to zero, then return AINT(*X*-0.5).

*Example:*

```
program test_anint
  real(4) x4
  real(8) x8
  x4 = 1.234E0_4
  x8 = 4.321_8
  print *, anint(x4), dnint(x8)
  x8 = anint(x4,8)
end program test_anint
```

*Specific names:*

Name	Argument	Return type	Option
DNINT( <i>X</i> )	REAL(8) <i>X</i>	REAL(8)	f95, gnu

## 7.13 ANY — Any value in MASK along DIM is true

*Description:*

ANY(MASK [, *DIM*]) determines if any of the values is true in *MASK* in the array along dimension *DIM*.

*Option:* f95, gnu

*Type:* transformational function

*Syntax:* `L = ANY(MASK)`  
`L = ANY(MASK, DIM)`

*Arguments:*

*MASK* The type of the argument shall be `LOGICAL(*)` and it shall not be scalar.

*DIM* (Optional) *DIM* shall be a scalar integer with a value that lies between one and the rank of *MASK*.

*Return value:*

`ANY(MASK)` returns a scalar value of type `LOGICAL(*)` where the kind type parameter is the same as the kind type parameter of *MASK*. If *DIM* is present, then `ANY(MASK, DIM)` returns an array with the rank of *MASK* minus 1. The shape is determined from the shape of *MASK* where the *DIM* dimension is elided.

- (A) `ANY(MASK)` is true if any element of *MASK* is true; otherwise, it is false. It also is false if *MASK* has zero size.
- (B) If the rank of *MASK* is one, then `ANY(MASK, DIM)` is equivalent to `ANY(MASK)`. If the rank is greater than one, then `ANY(MASK, DIM)` is determined by applying `ANY` to the array sections.

*Example:*

```

program test_any
  logical l
  l = any((/.true., .true., .true./))
  print *, l
  call section
contains
  subroutine section
    integer a(2,3), b(2,3)
    a = 1
    b = 1
    b(2,2) = 2
    print *, any(a .eq. b, 1)
    print *, any(a .eq. b, 2)
  end subroutine section
end program test_any

```

## 7.14 ASIN — Arcsine function

*Description:*

`ASIN(X)` computes the arcsine of its *X*.

*Option:* `f95, gnu`

*Type:* elemental function

*Syntax:* `X = ASIN(X)`

*Arguments:*

*X* The type shall be an `REAL(*)`, and a magnitude that is less than one.

*Return value:*

The return value is of type `REAL(*)` and it lies in the range  $\pi/2 \leq \arccos(x) \leq \pi/2$ . The kind type parameter is the same as *X*.

*Example:*

```
program test_asin
  real(8) :: x = 0.866_8
  x = asin(x)
end program test_asin
```

*Specific names:*

Name	Argument	Return type	Option
DASIN(X)	REAL(8) X	REAL(8)	f95, gnu

## 7.15 ATAN — Arctangent function

*Description:*

ATAN(X) computes the arctangent of *X*.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* X = ATAN(X)

*Arguments:*

X                      The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)` and it lies in the range  $-\pi/2 \leq \arcsin(x) \leq \pi/2$ .

*Example:*

```
program test_atan
  real(8) :: x = 2.866_8
  x = atan(x)
end program test_atan
```

*Specific names:*

Name	Argument	Return type	Option
DATAN(X)	REAL(8) X	REAL(8)	f95, gnu

## 7.16 BESJ0 — Bessel function of the first kind of order 0

*Description:*

BESJ0(X) computes the Bessel function of the first kind of order 0 of *X*.

*Option:* gnu

*Type:* elemental function

*Syntax:* X = BESJ0(X)

*Arguments:*

X                      The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)` and it lies in the range  $-0.4027... \leq Bessel(0, x) \leq 1$ .

*Example:*

```
program test_besj0
  real(8) :: x = 0.0_8
  x = besj0(x)
end program test_besj0
```

*Specific names:*

Name	Argument	Return type	Option
DBESJ0(X)	REAL(8) X	REAL(8)	gnu

## 7.17 BESJ1 — Bessel function of the first kind of order 1

*Description:*

BESJ1(X) computes the Bessel function of the first kind of order 1 of X.

*Option:* gnu

*Type:* elemental function

*Syntax:* X = BESJ1(X)

*Arguments:*

X                      The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)` and it lies in the range  $-0.5818... \leq Bessel(0, x) \leq 0.5818$ .

*Example:*

```
program test_besj1
  real(8) :: x = 1.0_8
  x = besj1(x)
end program test_besj1
```

*Specific names:*

Name	Argument	Return type	Option
DBESJ1(X)	REAL(8) X	REAL(8)	gnu

## 7.18 BESJN — Bessel function of the first kind

*Description:*

BESJN(N, X) computes the Bessel function of the first kind of order N of X.

*Option:* gnu

*Type:* elemental function

*Syntax:* Y = BESJN(N, X)

*Arguments:*

N                      The type shall be an `INTEGER(*)`.  
 X                      The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)`.

*Example:*

```
program test_besjn
  real(8) :: x = 1.0_8
  x = besjn(5,x)
end program test_besjn
```

*Specific names:*

Name	Argument	Return type	Option
DBESJN(X)	REAL(8) X	REAL(8)	gnu

## 7.19 BESY0 — Bessel function of the second kind of order 0

*Description:*

BESY0(X) computes the Bessel function of the second kind of order 0 of X.

*Option:* gnu

*Type:* elemental function

*Syntax:* X = BESY0(X)

*Arguments:*

X                      The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)`.

*Example:*

```
program test_besy0
  real(8) :: x = 0.0_8
  x = besy0(x)
end program test_besy0
```

*Specific names:*

Name	Argument	Return type	Option
DBESY0(X)	REAL(8) X	REAL(8)	gnu

## 7.20 BESY1 — Bessel function of the second kind of order 1

*Description:*

BESY1(X) computes the Bessel function of the second kind of order 1 of X.

*Option:* gnu

*Type:* elemental function

*Syntax:* X = BESY1(X)

*Arguments:*

X                      The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)`.

*Example:*

```

program test_besy1
  real(8) :: x = 1.0_8
  x = besy1(x)
end program test_besy1

```

*Specific names:*

Name	Argument	Return type	Option
DBESY1(X)	REAL(8) X	REAL(8)	gnu

## 7.21 BESYN — Bessel function of the second kind

*Description:*

BESYN(N, X) computes the Bessel function of the second kind of order  $N$  of  $X$ .

*Option:* gnu

*Type:* elemental function

*Syntax:* Y = BESYN(N, X)

*Arguments:*

N	The type shall be an INTEGER(*).
X	The type shall be an REAL(*).

*Return value:*

The return value is of type REAL(\*).

*Example:*

```

program test_besyn
  real(8) :: x = 1.0_8
  x = besyn(5,x)
end program test_besyn

```

*Specific names:*

Name	Argument	Return type	Option
DBESYN(X)	REAL(8) X	REAL(8)	gnu

## 7.22 COS — Cosine function

*Description:*

COS(X) computes the cosine of  $X$ .

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* X = COS(X)

*Arguments:*

X	The type shall be an REAL(*) or COMPLEX(*).
---	---

*Return value:*

The return value has same type and kind than  $X$ .

*Example:*



```

program test_cos
  real :: x = 0.0
  x = cos(x)
end program test_cos

```

*Specific names:*

Name	Argument	Return type	Option
DCOS(X)	REAL(8) X	REAL(8)	f95, gnu
CCOS(X)	COMPLEX(4) X	COMPLEX(4)	f95, gnu
ZCOS(X)	COMPLEX(8) X	COMPLEX(8)	f95, gnu
CDCOS(X)	COMPLEX(8) X	COMPLEX(8)	f95, gnu

## 7.23 COSH — Hyperbolic cosine function

*Description:*

COSH(X) computes the hyperbolic cosine of X.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* X = COSH(X)

*Arguments:*

X                      The type shall be an REAL(\*).

*Return value:*

The return value is of type REAL(\*) and it is positive ( $\cosh(x) \geq 0$ ).

*Example:*

```

program test_cosh
  real(8) :: x = 1.0_8
  x = cosh(x)
end program test_cosh

```

*Specific names:*

Name	Argument	Return type	Option
DCOSH(X)	REAL(8) X	REAL(8)	f95, gnu

## 7.24 ERF — Error function

*Description:*

ERF(X) computes the error function of X.

*Option:* gnu

*Type:* elemental function

*Syntax:* X = ERF(X)

*Arguments:*

X                      The type shall be an REAL(\*).

*Return value:*

The return value is of type REAL(\*) and it is positive ( $-1 \leq \operatorname{erf}(x) \leq 1$ ).

*Example:*

```

program test_erf
  real(8) :: x = 0.17_8
  x = erf(x)
end program test_erf

```

*Specific names:*

Name	Argument	Return type	Option
DERF(X)	REAL(8) X	REAL(8)	gnu

## 7.25 ERFC — Error function

*Description:*

ERFC(X) computes the complementary error function of X.

*Option:* gnu

*Type:* elemental function

*Syntax:* X = ERFC(X)

*Arguments:*

X                      The type shall be an REAL(\*).

*Return value:*

The return value is of type REAL(\*) and it is positive ( $0 \leq \text{erfc}(x) \leq 2$ ).

*Example:*

```

program test_erfc
  real(8) :: x = 0.17_8
  x = erfc(x)
end program test_erfc

```

*Specific names:*

Name	Argument	Return type	Option
DERFC(X)	REAL(8) X	REAL(8)	gnu

## 7.26 EXP — Exponential function

*Description:*

EXP(X) computes the base  $e$  exponential of X.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* X = EXP(X)

*Arguments:*

X                      The type shall be an REAL(\*) or COMPLEX(\*).

*Return value:*

The return value has same type and kind than X.

*Example:*

```

program test_exp
  real :: x = 1.0
  x = exp(x)
end program test_exp

```

*Specific names:*

Name	Argument	Return type	Option
DEXP(X)	REAL(8) X	REAL(8)	f95, gnu
CEXP(X)	COMPLEX(4) X	COMPLEX(4)	f95, gnu
ZEXP(X)	COMPLEX(8) X	COMPLEX(8)	f95, gnu
CDEXP(X)	COMPLEX(8) X	COMPLEX(8)	f95, gnu

## 7.27 LOG — Logarithm function

*Description:*

LOG(X) computes the logarithm of X.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* X = LOG(X)

*Arguments:*

X                      The type shall be an REAL(\*) or COMPLEX(\*).

*Return value:*

The return value is of type REAL(\*) or COMPLEX(\*). The kind type parameter is the same as X.

*Example:*

```

program test_log
  real(8) :: x = 1.0_8
  complex :: z = (1.0, 2.0)
  x = log(x)
  z = log(z)
end program test_log

```

*Specific names:*

Name	Argument	Return type	Option
ALOG(X)	REAL(4) X	REAL(4)	f95, gnu
DLOG(X)	REAL(8) X	REAL(8)	f95, gnu
CLOG(X)	COMPLEX(4) X	COMPLEX(4)	f95, gnu
ZLOG(X)	COMPLEX(8) X	COMPLEX(8)	f95, gnu
CDLOG(X)	COMPLEX(8) X	COMPLEX(8)	f95, gnu

## 7.28 LOG10 — Base 10 logarithm function

*Description:*

LOG10(X) computes the base 10 logarithm of X.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* X = LOG10(X)

*Arguments:*

X                      The type shall be an REAL(\*) or COMPLEX(\*).

*Return value:*

The return value is of type `REAL(*)` or `COMPLEX(*)`. The kind type parameter is the same as `X`.

*Example:*

```
program test_log10
  real(8) :: x = 10.0_8
  x = log10(x)
end program test_log10
```

*Specific names:*

Name	Argument	Return type	Option
<code>ALOG10(X)</code>	<code>REAL(4) X</code>	<code>REAL(4)</code>	f95, gnu
<code>DLOG10(X)</code>	<code>REAL(8) X</code>	<code>REAL(8)</code>	f95, gnu

**7.29 SIN — Sine function***Description:*

`SIN(X)` computes the sine of `X`.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* `X = SIN(X)`

*Arguments:*

`X` The type shall be an `REAL(*)` or `COMPLEX(*)`.

*Return value:*

The return value has same type and kind than `X`.

*Example:*

```
program test_sin
  real :: x = 0.0
  x = sin(x)
end program test_sin
```

*Specific names:*

Name	Argument	Return type	Option
<code>DSIN(X)</code>	<code>REAL(8) X</code>	<code>REAL(8)</code>	f95, gnu
<code>CSIN(X)</code>	<code>COMPLEX(4) X</code>	<code>COMPLEX(4)</code>	f95, gnu
<code>ZSIN(X)</code>	<code>COMPLEX(8) X</code>	<code>COMPLEX(8)</code>	f95, gnu
<code>CDSIN(X)</code>	<code>COMPLEX(8) X</code>	<code>COMPLEX(8)</code>	f95, gnu

**7.30 SINH — Hyperbolic sine function***Description:*

`SINH(X)` computes the hyperbolic sine of `X`.

*Option:* f95, gnu

*Type:* elemental function

*Syntax:* `X = SINH(X)`

*Arguments:*

*X*                      The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)`.

*Example:*

```
program test_sinh
  real(8) :: x = - 1.0_8
  x = sinh(x)
end program test_sinh
```

*Specific names:*

Name	Argument	Return type	Option
<code>DSINH(X)</code>	<code>REAL(8) X</code>	<code>REAL(8)</code>	f95, gnu

**7.31 SQRT — Square-root function***Description:*

`SQRT(X)` computes the square root of *X*.

*Option:*    f95, gnu

*Type:*     elemental function

*Syntax:*    `X = SQRT(X)`

*Arguments:*

*X*                      The type shall be an `REAL(*)` or `COMPLEX(*)`.

*Return value:*

The return value is of type `REAL(*)` or `COMPLEX(*)`. The kind type parameter is the same as *X*.

*Example:*

```
program test_sqrt
  real(8) :: x = 2.0_8
  complex :: z = (1.0, 2.0)
  x = sqrt(x)
  z = sqrt(z)
end program test_sqrt
```

*Specific names:*

Name	Argument	Return type	Option
<code>DSQRT(X)</code>	<code>REAL(8) X</code>	<code>REAL(8)</code>	f95, gnu
<code>CSQRT(X)</code>	<code>COMPLEX(4) X</code>	<code>COMPLEX(4)</code>	f95, gnu
<code>ZSQRT(X)</code>	<code>COMPLEX(8) X</code>	<code>COMPLEX(8)</code>	f95, gnu
<code>CDSQRT(X)</code>	<code>COMPLEX(8) X</code>	<code>COMPLEX(8)</code>	f95, gnu

**7.32 TAN — Tangent function***Description:*

`TAN(X)` computes the tangent of *X*.

*Option:*    f95, gnu

*Type:* elemental function

*Syntax:*  $X = \text{TAN}(X)$

*Arguments:*

$X$  The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)`. The kind type parameter is the same as  $X$ .

*Example:*

```
program test_tan
  real(8) :: x = 0.165_8
  x = tan(x)
end program test_tan
```

*Specific names:*

Name	Argument	Return type	Option
DTAN( $X$ )	<code>REAL(8)</code> $X$	<code>REAL(8)</code>	f95, gnu

### 7.33 TANH — Hyperbolic tangent function

*Description:*

`TANH( $X$ )` computes the hyperbolic tangent of  $X$ .

*Option:* f95, gnu

*Type:* elemental function

*Syntax:*  $X = \text{TANH}(X)$

*Arguments:*

$X$  The type shall be an `REAL(*)`.

*Return value:*

The return value is of type `REAL(*)` and lies in the range  $-1 \leq \tanh(x) \leq 1$ .

*Example:*

```
program test_tanh
  real(8) :: x = 2.1_8
  x = tanh(x)
end program test_tanh
```

*Specific names:*

Name	Argument	Return type	Option
DTANH( $X$ )	<code>REAL(8)</code> $X$	<code>REAL(8)</code>	f95, gnu

## 8 Contributing

Free software is only possible if people contribute to efforts to create it. We're always in need of more people helping out with ideas and comments, writing documentation and contributing code.

If you want to contribute to GNU Fortran 95, have a look at the long lists of projects you can take on. Some of these projects are small, some of them are large; some are completely orthogonal to the rest of what is happening on **gfortran**, but others are “mainstream” projects in need of enthusiastic hackers. All of these projects are important! We'll eventually get around to the things here, but they are also things doable by someone who is willing and able.

### 8.1 Contributors to GNU Fortran 95

Most of the parser was hand-crafted by *Andy Vaught*, who is also the initiator of the whole project. Thanks Andy! Most of the interface with GCC was written by *Paul Brook*.

The following individuals have contributed code and/or ideas and significant help to the **gfortran** project (in no particular order):

- Andy Vaught
- Katherine Holcomb
- Tobias Schlter
- Steven Bosscher
- Toon Moene
- Tim Prince
- Niels Kristian Bech Jensen
- Steven Johnson
- Paul Brook
- Feng Wang
- Bud Davis

The following people have contributed bug reports, smaller or larger patches, and much needed feedback and encouragement for the **gfortran** project:

- Erik Schnetter
- Bill Clodius
- Kate Hedstrom

Many other individuals have helped debug, test and improve **gfortran** over the past two years, and we welcome you to do the same! If you already have done so, and you would like to see your name listed in the list above, please contact us.

### 8.2 Projects

*Help build the test suite*

Solicit more code for donation to the test suite. We can keep code private on request.

*Bug hunting/squishing*

Find bugs and write more test cases! Test cases are especially very welcome, because it allows us to concentrate on fixing bugs instead of isolating them.

*Smaller projects (“bug” fixes):*

- Allow init exprs to be numbers raised to integer powers.
- Implement correct rounding.
- Implement F restrictions on Fortran 95 syntax.
- See about making Emacs-parsable error messages.

If you wish to work on the runtime libraries, please contact a project maintainer.



## 9 Standards

The GNU Fortran 95 Compiler aims to be a conforming implementation of ISO/IEC 1539:1997 (Fortran 95).

In the future it may also support other variants and extensions to the Fortran language. This includes ANSI Fortran 77, Fortran 90, Fortran 2000 (not yet finalized), and OpenMP.



# Index

## -

-fbounds-check option	30
-fdefault-double-8, option	26
-fdefault-integer-8, option	26
-fdefault-real-8, option	26
-fdollar-ok option	26
-fdump-parse-tree option	28
-fixed-line-length- <i>n</i> option	26
-ffortran-bounds-check option	30
-ffree-form option	26
-fimplicit-none option	26
-fmax-identifier-length= <i>n</i> option	26
-fmax-stack-var-size option	30
-fno-fixed-form option	26
-fno-second-underscore option	30
-fno-underscoring option	29
-fpackderived	30
-frepack-arrays option	30
-fsyntax-only option	27
-Idir option	28
-Mdir option	29
-pedantic option	27
-pedantic-errors option	27
-std= <i>std</i> option	26
-w option	27
-W option	28
-Waliasing option	27
-Wall option	27
-Wconversion option	27
-Werror	28
-Wimplicit-interface option	27
-Wnonstd-intrinsic option	27
-Wsurprising	28
-Wunderflow	28
-Wunused-labels option	28

## A

abort	39
ABORT	39
ABS intrinsic	40
absolute value	40
ACHAR intrinsic	40
ACOS intrinsic	41
adjust string	41, 42
ADJUSTL intrinsic	41
ADJUSTR intrinsic	42
AIMAG intrinsic	42
AINTR intrinsic	43
aliasing	27
ALL intrinsic	43
all warnings	27
ALLOCATED intrinsic	44
allocation status	44

ALOG intrinsic	53
ALOG10 intrinsic	53
ANINT intrinsic	45
ANY intrinsic	45
arccosine	41
arcsine	46
arctangent	47
array bounds checking	30
ASCII collating sequence	40
ASIN intrinsic	46
ATAN intrinsic	47
Authors	57

## B

BESJ0 intrinsic	47
BESJ1 intrinsic	48
BESJN intrinsic	48
Bessel	47, 48, 49, 50
BESY0 intrinsic	49
BESY1 intrinsic	49
BESYN intrinsic	50
bounds checking	30

## C

CABS intrinsic	40
card image	26
CDABS intrinsic	40
CDCOS intrinsic	50
CDEXP intrinsic	52
CDLOG intrinsic	53
CDSIN intrinsic	54
CDSQRT intrinsic	55
character set	26
checking subscripts	30
CLOG intrinsic	53
code generation, conventions	29
command options	25
Contributing	57
Contributors	57
conversion	27
COS intrinsic	50
COSH intrinsic	51
cosine	50
Credits	57
CSQRT intrinsic	55

## D

DABS intrinsic	40
DACOS intrinsic	41
DASIN intrinsic	46
DATAN intrinsic	47

DBESJ0 intrinsic .....	47
DBESJ1 intrinsic .....	48
DBESJN intrinsic .....	48
DBESY0 intrinsic .....	49
DBESY1 intrinsic .....	49
DBESYN intrinsic .....	50
DCOS intrinsic .....	50
DCOSH intrinsic .....	51
debugging information options .....	28
DEXP intrinsic .....	52
dialect options .....	26
DIMAG intrinsic .....	42
DINT intrinsic .....	43
directive, INCLUDE .....	28
directory, options .....	28
directory, search paths for inclusion .....	28
DLOG intrinsic .....	53
DLOG10 intrinsic .....	53
DNINT intrinsic .....	45
dollar sign .....	26
DSIN intrinsic .....	54
DSINH intrinsic .....	54
DSQRT intrinsic .....	55
DTAN intrinsic .....	55
DTANH intrinsic .....	56

## E

environment variables .....	31
ERF intrinsic .....	51
ERFC intrinsic .....	52
error .....	51, 52
EXP intrinsic .....	52
exponential .....	52
extended-source option .....	26
Extension .....	37
extra warnings .....	28

## F

FDL, GNU Free Documentation License .....	9
fixed form .....	26
Fortran 77 .....	23
Fortran 90, features .....	26
free form .....	26

## G

G77 .....	23
GNU Compiler Collection .....	21
GNU Fortran 95 command options .....	25

## H

hyperbolic cosine .....	51
hyperbolic sine .....	54
hyperbolic tangent .....	56

## I

IABS intrinsic .....	40
Imaginary part .....	42
INCLUDE directive .....	28
inclusion, directory search paths for .....	28
Initialization .....	37
Intrinsic Procedures .....	39
Introduction .....	1

## K

Kind specifications .....	37
---------------------------	----

## L

labels, unused .....	28
language, dialect options .....	26
length of source lines .....	26
limits, lengths of source lines .....	26
lines, length .....	26
LOG intrinsic .....	53
LOG10 intrinsic .....	53
logarithm .....	53

## M

messages, warning .....	27
module search path .....	28

## N

negative forms of options .....	25
---------------------------------	----

## O

option -fmax-identifier-length= <i>n</i> .....	26
option, -fdefault-double-8 .....	26
option, -fdefault-integer-8 .....	26
option, -fdefault-real-8 .....	26
option, -fdump-parse-tree .....	28
option, -Mdir .....	29
option, -std= <i>std</i> .....	26
options, -fdollar-ok .....	26
options, -ffixed-line-length- <i>n</i> .....	26
options, -ffree-form .....	26
options, -fimplicit-none .....	26
options, -fno-fixed-form .....	26
options, -fno-second-underscore .....	30
options, -fno-underscoring .....	29
options, -fsyntax-only .....	27
options, -Idir .....	28
options, -pedantic .....	27
options, -pedantic-errors .....	27
options, -w .....	27
options, -W .....	28
options, -Wall .....	27
options, -Wall .....	27

options, -Wconversion .....	27
options, -Werror .....	28
options, -Wimplicit-interface .....	27
options, -Wnonstd-intrinsic .....	27
options, -Wsurprising .....	28
options, -Wunderflow .....	28
options, -Wunused-labels .....	28
options, code generation .....	29
options, debugging .....	28
options, dialect .....	26
options, directory search .....	28
options, GNU Fortran 95 command .....	25
options, negative forms .....	25
options, warnings .....	27

## P

paths, search .....	28
---------------------	----

## R

range checking .....	30
Repacking arrays .....	30
run-time, options .....	29

## S

search path .....	28
search paths, for included files .....	28
<b>SIN intrinsic</b> .....	54
sine .....	54
<b>SINH intrinsic</b> .....	54
source file format .....	26
Source Form .....	26
<b>SQRT intrinsic</b> .....	55
square-root .....	55
Standards .....	59
Structure packing .....	30

subscript checking .....	30
suppressing warnings .....	27
Suspicious .....	28
symbol names .....	26
symbol names, transforming .....	29, 30
symbol names, underscores .....	29, 30
syntax checking .....	27

## T

<b>TAN intrinsic</b> .....	55
tangent .....	55
<b>TANH intrinsic</b> .....	56
transforming symbol names .....	29, 30
true values .....	43, 45

## U

<b>UNDERFLOW</b> .....	28
underscore .....	29, 30
unused labels .....	28

## W

warnings, all .....	27
warnings, extra .....	28
warnings, suppressing .....	27
whole number .....	43, 45

## Z

<b>ZABS intrinsic</b> .....	40
<b>ZCOS intrinsic</b> .....	50
<b>ZEXP intrinsic</b> .....	52
<b>ZLOG intrinsic</b> .....	53
<b>ZSIN intrinsic</b> .....	54
<b>ZSQRT intrinsic</b> .....	55

