



# Sun Studio 12: Fortran Library Reference



Sun Microsystems, Inc.  
4150 Network Circle  
Santa Clara, CA 95054  
U.S.A.

Part No: 819-5259

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

---

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux États-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certains composants de ce produit peuvent être dérivés du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux États-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des États-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

# Contents

---

<b>Preface</b> .....	7
<b>1 Fortran Library Routines</b> .....	13
1.1 Data Type Considerations .....	13
1.2 64-Bit Environments .....	14
1.3 Fortran Math Functions .....	15
1.3.1 Single-Precision Functions .....	15
1.3.2 Double-Precision Functions .....	19
1.3.3 Quad-Precision Functions .....	23
1.4 Fortran Library Routines Reference .....	24
1.4.1 <b>abort</b> : Terminate and Write Core File .....	25
1.4.2 <b>access</b> : Check File Permissions or Existence .....	25
1.4.3 <b>alarm</b> : Call Subroutine After a Specified Time .....	26
1.4.4 <b>bit</b> : Bit Functions: <b>and</b> , <b>or</b> , ..., <b>bit</b> , <b>setbit</b> , ... .....	27
1.4.5 <b>chdir</b> : Change Default Directory .....	30
1.4.6 <b>chmod</b> : Change the Mode of a File .....	30
1.4.7 <b>date</b> : Get Current Date as a Character String .....	31
1.4.8 <b>dtime</b> , <b>etime</b> : Elapsed Execution Time .....	34
1.4.9 <b>exit</b> : Terminate a Process and Set the Status .....	36
1.4.10 <b>fdate</b> : Return Date and Time in an ASCII String .....	36
1.4.11 <b>flush</b> : Flush Output to a Logical Unit .....	37
1.4.12 <b>fork</b> : Create a Copy of the Current Process .....	38
1.4.13 <b>fseek</b> , <b>ftell</b> : Determine Position and Reposition a File .....	38
1.4.14 <b>fseeko64</b> , <b>ftello64</b> : Determine Position and Reposition a Large File .....	40
1.4.15 <b>getarg</b> , <b>iargc</b> : Get Command-Line Arguments .....	42
1.4.16 <b>getc</b> , <b>fgetc</b> : Get Next Character .....	43
1.4.17 <b>getcwd</b> : Get Path of Current Working Directory .....	45
1.4.18 <b>getenv</b> : Get Value of Environment Variables .....	45

---

1.4.19 <b>getfd</b> : Get File Descriptor for External Unit Number .....	46
1.4.20 <b>getfilep</b> : Get File Pointer for External Unit Number .....	46
1.4.21 <b>getlog</b> : Get User's Login Name .....	48
1.4.22 <b>getpid</b> : Get Process ID .....	48
1.4.23 <b>getuid, getgid</b> : Get User or Group ID of Process .....	48
1.4.24 <b>hostname</b> : Get Name of Current Host .....	49
1.4.25 <b>idate</b> : Return Current Date .....	50
1.4.26 <b>ieee_flags, ieee_handler, sigfpe</b> : IEEE Arithmetic .....	50
1.4.27 <b>index, rindex, lnlbnk</b> : Index or Length of Substring .....	55
1.4.28 <b>inmax</b> : Return Maximum Positive Integer .....	57
1.4.29 <b>itime</b> : Current Time .....	58
1.4.30 <b>kill</b> : Send a Signal to a Process .....	58
1.4.31 <b>link, symlink</b> : Make a Link to an Existing File .....	59
1.4.32 <b>loc</b> : Return the Address of an Object .....	60
1.4.33 <b>long, short</b> : Integer Object Conversion .....	61
1.4.34 <b>longjmp, issetjmp</b> : Return to Location Set by <b>issetjmp</b> .....	62
1.4.35 <b>malloc, malloc64, realloc, free</b> : Allocate/Reallocate/Deallocate Memory .....	63
1.4.36 <b>mvbits</b> : Move a Bit Field .....	66
1.4.37 <b>perror, gerror, ierrno</b> : Get System Error Messages .....	67
1.4.38 <b>putc, fputc</b> : Write a Character to a Logical Unit .....	69
1.4.39 <b>qsort, qsort64</b> : Sort the Elements of a One-Dimensional Array .....	71
1.4.40 <b>ran</b> : Generate a Random Number Between 0 and 1 .....	72
1.4.41 <b>rand, drand, irand</b> : Return Random Values .....	73
1.4.42 <b>rename</b> : Rename a File .....	74
1.4.43 <b>secsds</b> : Get System Time in Seconds, Minus Argument .....	75
1.4.44 <b>set_io_err_handler, get_io_err_handler</b> : Set and Get I/O Error Handler .....	76
1.4.45 <b>sh</b> : Fast Execution of an <b>sh</b> Command .....	79
1.4.46 <b>signal</b> : Change the Action for a Signal .....	80
1.4.47 <b>sleep</b> : Suspend Execution for an Interval .....	81
1.4.48 <b>stat, lstat, fstat</b> : Get File Status .....	81
1.4.49 <b>stat64, lstat64, fstat64</b> : Get File Status .....	84
1.4.50 <b>system</b> : Execute a System Command .....	84
1.4.51 <b>time, ctime, ltime, gmtime</b> : Get System Time .....	85
1.4.52 <b>ttynam, isatty</b> : Get Name of a Terminal Port .....	89
1.4.53 <b>unlink</b> : Remove a File .....	90
1.4.54 <b>wait</b> : Wait for a Process to Terminate .....	90

---

<b>2 Fortran 95 Intrinsic Functions</b> .....	93
2.1 Standard Fortran 95 Generic Intrinsic Functions .....	93
2.1.1 Argument Presence Inquiry Function .....	93
2.1.2 Numeric Functions .....	93
2.1.3 Mathematical Functions .....	94
2.1.4 Character Functions .....	95
2.1.5 Character Inquiry Function .....	96
2.1.6 Kind Functions .....	96
2.1.7 Logical Function .....	96
2.1.8 Numeric Inquiry Functions .....	96
2.1.9 Bit Inquiry Function .....	97
2.1.10 Bit Manipulation Functions .....	97
2.1.11 Transfer Function .....	98
2.1.12 Floating-Point Manipulation Functions .....	98
2.1.13 Vector and Matrix Multiply Functions .....	98
2.1.14 Array Reduction Functions .....	98
2.1.15 Array Inquiry Functions .....	99
2.1.16 Array Construction Functions .....	99
2.1.17 Array Reshape Function .....	100
2.1.18 Array Manipulation Functions .....	100
2.1.19 Array Location Functions .....	100
2.1.20 Pointer Association Status Functions .....	100
2.1.21 System Environment Procedures .....	101
2.1.22 Intrinsic Subroutines .....	101
2.1.23 Specific Names for Intrinsic Functions .....	102
2.2 Fortran 2003 Module Routines .....	105
2.2.1 IEEE Arithmetic and Exceptions Modules .....	105
2.2.2 C Binding Module .....	108
2.3 Non-Standard Fortran 95 Intrinsic Functions .....	108
2.3.1 Basic Linear Algebra Functions (BLAS) .....	108
2.3.2 Interval Arithmetic Intrinsic Functions .....	109
2.3.3 Other Vendor Intrinsic Functions .....	110
2.3.4 Other Extensions .....	111

---

<b>3 FORTRAN 77 and VMS Intrinsic Functions</b> .....	113
3.1 Arithmetic and Mathematical Functions .....	114
3.1.1 Arithmetic Functions .....	114
3.1.2 Type Conversion Functions .....	116
3.1.3 Trigonometric Functions .....	120
3.1.4 Other Mathematical Functions .....	122
3.2 Character Functions .....	124
3.3 Miscellaneous Functions .....	125
3.3.1 Bit Manipulation <sup>⌘</sup> .....	125
3.3.2 Environmental Inquiry Functions <sup>⌘</sup> .....	125
3.3.3 Memory <sup>⌘</sup> .....	126
3.4 Remarks .....	127
3.4.1 Notes on Functions .....	128
3.5 VMS Intrinsic Functions .....	132
3.5.1 VMS Double-Precision Complex .....	133
3.5.2 VMS Degree-Based Trigonometric .....	133
3.5.3 VMS Bit-Manipulation .....	134
3.5.4 VMS Multiple Integer Types .....	136
<b>Index</b> .....	137

# Preface

---

The *Fortran Library Reference* describes the intrinsic functions and routines in the Sun™ Studio Fortran libraries. This reference manual is intended for programmers with a working knowledge of the Fortran language and the Solaris™ operating environment.

This guide is intended for scientists, engineers, and programmers who have a working knowledge of the Fortran language and wish to learn how to use the Sun Fortran compilers effectively. Familiarity with the Solaris operating environment or UNIX® in general is also assumed.

Discussion of Fortran programming issues on Solaris operating environments, including input/output, application development, library creation and use, program analysis, porting, optimization, and parallelization can be found in the companion *Fortran Programming Guide*.

## Typographic Conventions

TABLE P-1 Typeface Conventions

Typeface	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail.
<b>AaBbCc123</b>	What you type, when contrasted with on-screen computer output	% <b>su</b> Password:
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be superuser to do this.
<code>AaBbCc123</code>	Command-line placeholder text; replace with a real name or value	To delete a file, type <code>rm filename</code> .

- The symbol ∇ stands for a blank space where a blank is significant:

VV36.001

- The FORTRAN 77 standard used an older convention, spelling the name “FORTRAN” capitalized. The current convention is to use lower case: “Fortran 95”
- References to online man pages appear with the topic name and section number. For example, a reference to the library routine GETENV will appear as `getenv(3F)`, implying that the man command to access this man page would be: `man -s 3F getenv`

TABLE P-2 Code Conventions

Code Symbol	Meaning	Notation	Code Example
[ ]	Brackets contain arguments that are optional.	$O[n]$	-O4, -O
{ }	Braces contain a set of choices for a required option.	$d\{y n\}$	-dy
	The “pipe” or “bar” symbol separates arguments, only one of which may be chosen.	$B\{\text{dynamic} \text{static}\}$	-Bstatic
:	The colon, like the comma, is sometimes used to separate arguments.	$Rdir[:dir]$	-R/local/libs:/U/a
...	The ellipsis indicates omission in a series.	$-xinline=f1[,...fn]$	-xinline=alpha,dos

## Shell Prompts

Shell	Prompt
C shell	<i>machine-name%</i>
C shell superuser	<i>machine-name#</i>
Bourne shell, Korn shell, and GNU Bourne-Again shell	\$
Superuser for Bourne shell, Korn shell, and GNU Bourne-Again shell	#



## Supported Platforms

This Sun Studio release supports systems that use the SPARC® and x86 families of processor architectures: UltraSPARC™, SPARC64, AMD64, Pentium, and Xeon EM64T. The supported systems for the version of the Solaris Operating System you are running are available in the hardware compatibility lists at <http://www.sun.com/bigadmin/hcl>. These documents cite any implementation differences between the platform types.

In this document, these x86 related terms mean the following:

- “x86” refers to the larger family of 64-bit and 32-bit x86 compatible products.
- “x64” points out specific 64-bit information about AMD64 or EM64T systems.
- “32-bit x86” points out specific 32-bit information about x86 based systems.

For supported systems, see the hardware compatibility lists.

## Accessing Sun Studio Documentation

You can access the documentation at the following locations:

- The documentation is available from the documentation index that is installed with the software on your local system or network at `file:/opt/SUNWspro/docs/index.html` on Solaris platforms and at `file:/opt/sun/sunstudio12/docs/index.html` on Linux platforms.

If your software is not installed in the `/opt` directory on a Solaris platform or the `/opt/sun` directory on a Linux platform, ask your system administrator for the equivalent path on your system.

- Most manuals are available from the `docs.sun.comsm` web site. The following titles are available through your installed software on Solaris platforms only:
  - *Standard C++ Library Class Reference*
  - *Standard C++ Library User's Guide*
  - *Tools.h++ Class Library Reference*
  - *Tools.h++ User's Guide*

The release notes for both Solaris platforms and Linux platforms are available from the `docs.sun.com` web site.

- Online help for all components of the IDE is available through the Help menu, as well as through Help buttons on many windows and dialog boxes, in the IDE.

The `docs.sun.com` web site (<http://docs.sun.com>) enables you to read, print, and buy Sun Microsystems manuals through the Internet. If you cannot find a manual, see the documentation index that is installed with the software on your local system or network.

---

**Note** – Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

---

## Documentation in Accessible Formats

The documentation is provided in accessible formats that are readable by assistive technologies for users with disabilities. You can find accessible versions of documentation as described in the following table. If your software is not installed in the /opt directory, ask your system administrator for the equivalent path on your system.

Type of Documentation	Format and Location of Accessible Version
Manuals (except third-party manuals)	HTML at <a href="http://docs.sun.com">http://docs.sun.com</a>
Third-party manuals: <ul style="list-style-type: none"> <li>▪ <i>Standard C++ Library Class Reference</i></li> <li>▪ <i>Standard C++ Library User's Guide</i></li> <li>▪ <i>Tools.h++ Class Library Reference</i></li> <li>▪ <i>Tools.h++ User's Guide</i></li> </ul>	HTML in the installed software on Solaris platforms through the documentation index at <code>file:/opt/SUNWspro/docs/index.html</code>
Readmes	HTML on the Sun Developer Network portal at <a href="http://developers.sun.com/sunstudio/documentation/ss12m">http://developers.sun.com/sunstudio/documentation/ss12m</a>
Man pages	HTML in the installed software through the documentation index at <code>file:/opt/SUNWspro/docs/index.html</code> on Solaris platforms, and at <code>file:/opt/sun/sunstudio12/docs/index.html</code> on Linux platforms,
Online help	HTML available through the Help menu and Help buttons in the IDE
Release notes	HTML at <a href="http://docs.sun.com">http://docs.sun.com</a>

## Related Sun Studio Documentation

The following table describes related documentation that is available at `file:/opt/SUNWspro/docs/index.html` and <http://docs.sun.com>. If your software is not installed in the `/opt` directory, ask your system administrator for the equivalent path on your system.

Document Title	Description
<i>Fortran Programming Guide</i>	Describes how to write effective Fortran programs on Solaris environments; input/output, libraries, performance, debugging, and parallelization.
<i>Fortran User's Guide</i>	Describes the compile-time environment and command-line options for the f95 compiler.
<i>OpenMP API User's Guide</i>	Summary of the OpenMP multiprocessing API, with specifics about the implementation.
<i>Numerical Computation Guide</i>	Describes issues regarding the numerical accuracy of floating-point computations.

## Accessing Related Solaris Documentation

The following table describes related documentation that is available through the `docs.sun.com` web site.

Document Collection	Document Title	Description
Solaris Reference Manual Collection	See the titles of man page sections.	Provides information about the Solaris operating system.
Solaris Software Developer Collection	<i>Linker and Libraries Guide</i>	Describes the operations of the Solaris link-editor and runtime linker.
Solaris Software Developer Collection	<i>Multithreaded Programming Guide</i>	Covers the POSIX and Solaris threads APIs, programming with synchronization objects, compiling multithreaded programs, and finding tools for multithreaded programs.

## Resources for Developers

Visit the Sun Developer Network Sun Studio portal at <http://developers.sun.com/sunstudio> to find these frequently updated resources:

- Articles on programming techniques and best practices
- Documentation of the software, as well as corrections to the documentation that is installed with your software
- Information on support levels
- User forums
- Downloadable code samples
- New technology previews

The Sun Studio portal is one of a number of additional resources for developers at the Sun Developer Network website, <http://developers.sun.com>.

## Contacting Sun Technical Support

If you have technical questions about this product that are not answered in this document, go to:

<http://www.sun.com/service/contacting>

## Sending Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Submit your comments to Sun at this URL:

<http://www.sun.com/hwdocs/feedback>

Please include the part number of the document in the subject line of your email. For example, the part number for this document is 819-5259-10.

# Fortran Library Routines

---

This chapter describes the Fortran library routines.

All the routines described in this chapter have corresponding man pages in section 3F of the man library. For example, `man -s 3F access` will display the man page entry for the library routine `access`.

This chapter does not describe the standard Fortran 95 intrinsic routines. See the relevant Fortran 95 standards documents for information on intrinsics.

See also the *Numerical Computation Guide* for additional math routines that are callable from Fortran and C. These include the standard math library routines in `libm` and `libsunmath` (see `Intro(3M)`), optimized versions of these libraries, the SPARC vector math library, `libmvec`, and others.

See “[2.3.4.2 Memory Functions](#)” on page 112 for details on Fortran 77 and VMS intrinsic functions implemented by the `f95` compiler.

## 1.1 Data Type Considerations

Unless otherwise indicated, the function routines listed here are not intrinsics. That means that the type of data a function returns may conflict with the implicit typing of the function name, and require explicit type declaration by the user. For example, `getpid()` returns `INTEGER*4` and would require an `INTEGER*4 getpid` declaration to ensure proper handling of the result. (Without explicit typing, a `REAL` result would be assumed by default because the function name starts with `g`.) As a reminder, explicit type statements appear in the function summaries for these routines.

Be aware that `IMPLICIT` statements and the `-dbl` and `-xtypemap` compiler options also alter the data typing of arguments and the treatment of return values. A mismatch between the expected and actual data types in calls to these library routines could cause unexpected behavior. Options `-xtypemap` and `-dbl` promote the data type of `INTEGER` functions to `INTEGER*8`, `REAL` functions

to **REAL\*8**, and **DOUBLE** functions to **REAL\*16**. To protect against these problems, function names and variables appearing in library calls should be explicitly typed with their expected sizes, as in:

```
integer*4 seed, getuid
real*4 ran
...
seed = 70198
val = getuid() + ran(seed)
...
```

Explicit typing in the example protects the library calls from any data type promotion when the **-xtypemap** and **-dbl** compiler options are used. Without explicit typing, these options could produce unexpected results. See the *Fortran User's Guide* and the **f95(1)** man page for details on these options.

The Fortran 95 compiler, **f95**, provides an include file, **system.inc**, that defines the interfaces for most non-intrinsic library routines. Include this file to insure that functions you call and their arguments are properly typed, especially when default data types are changed with **-xtypemap**.

```
include 'system.inc'
integer(4) mypid
mypid = getpid()
print *, mypid
```

You can catch many issues related to type mismatches over library calls by using the Fortran compilers' global program checking option, **-XList**. Global program checking by the **f95** compiler is described in the *Fortran User's Guide*, the *Fortran Programming Guide*, and the **f95(1)** man page.

## 1.2 64-Bit Environments

Compiling a program to run in a 64-bit operating environment (that is, compiling with **-m64**, and running the executable on a 64-bit enabled SPARC or x86 platform) changes the return values of certain functions. These are usually functions that interface standard system-level routines, such as **malloc(3F)** (see “[1.4.35 malloc, malloc64, realloc, free: Allocate/Reallocate/Deallocate Memory](#)” on page 63), and may take or return 32-bit or 64-bit values depending on the environment. To provide portability of code between 32-bit and 64-bit environments, 64-bit versions of these routines have been provided that always take and/or return 64-bit values. The following table identifies library routine provided for use in 64-bit environments:

TABLE 1-1 Library Routines for 64-bit Environments

Function	Description
<code>malloc64</code>	Allocate memory and return a pointer
<code>fseeko64</code>	Reposition a large file
<code>ftello64</code>	Determine position of a large file
<code>stat64</code> , <code>fstat64</code> , <code>lstat64</code>	Determine status of a file
<code>time64</code> , <code>ctime64</code> , <code>gmtime64</code> , <code>ltime64</code>	Dissect system time, convert to character
<code>qsort64</code>	Sort the elements of an array

## 1.3 Fortran Math Functions

The following functions and subroutines are part of the Fortran math libraries. They are available to all programs compiled with **f95**. These routines are non-intrinsics that take a specific data type as an argument and return the same. Non-intrinsics do have to be declared in the routine referencing them.

Many of these routines are "wrappers", Fortran interfaces to routines in the C language library, and as such are non-standard Fortran. They include IEEE recommended support functions, and specialized random number generators. See the *Numerical Computation Guide* and the man pages `libm_single(3F)`, `libm_double(3F)`, `libm_quadruple(3F)`, for more information about these libraries.

### 1.3.1 Single-Precision Functions

These subprograms are single-precision math functions and subroutines.

In general, the functions below provide access to single-precision math functions that do *not* correspond to standard Fortran generic intrinsic functions—data types are determined by the usual data typing rules.

These functions need not be explicitly typed with a **REAL** statement as long as default typing holds. (Names beginning with “**r**” are **REAL**, with “**i**” are **INTEGER**.)

For details on these routines, see the C math library man pages (3M). For example, for `r_acos(x)` see the `acos(3M)` man page.

TABLE 1-2 Single-Precision Math Functions

Function Name	Return Type	Description
<code>r_acos( x )</code>	REAL	arc cosine
<code>r_acosd( x )</code>	REAL	--
<code>r_acosh( x )</code>	REAL	arc cosh
<code>r_acosp( x )</code>	REAL	--
<code>r_acospi( x )</code>	REAL	--
<code>r_atan( x )</code>	REAL	arc tangent
<code>r_atand( x )</code>	REAL	--
<code>r_atanh( x )</code>	REAL	arc tanh
<code>r_atanp( x )</code>	REAL	--
<code>r_atanpi( x )</code>	REAL	--
<code>r_asin( x )</code>	REAL	arc sine
<code>r_asind( x )</code>	REAL	--
<code>r_asinh( x )</code>	REAL	arc sinh
<code>r_asinp( x )</code>	REAL	--
<code>r_asinpi( x )</code>	REAL	--
<code>r_atan2(( y, x )</code>	REAL	arc tangent
<code>r_atan2d( y, x )</code>	REAL	--
<code>r_atan2pi( y, x )</code>	REAL	--
<code>r_cbrt( x )</code>	REAL	cube root
<code>r_ceil( x )</code>	REAL	ceiling
<code>r_copysign( x, y )</code>	REAL	--
<code>r_cos( x )</code>	REAL	cosine
<code>r_cosd( x )</code>	REAL	--
<code>r_cosh( x )</code>	REAL	hyperb cos
<code>r_cosp( x )</code>	REAL	--
<code>r_cospi( x )</code>	REAL	--
<code>r_erf( x )</code>	REAL	err function
<code>r_erfc( x )</code>	REAL	--



TABLE 1-2 Single-Precision Math Functions (Continued)

Function Name	Return Type	Description
<code>r_expm1( x )</code>	REAL	$(e^{**x}) - 1$
<code>r_floor( x )</code>	REAL	<b>f</b> loor
<code>r_hypot( x, y )</code>	REAL	<b>h</b> ypotenuse
<code>r_infinity( )</code>	REAL	--
<code>r_j0( x )</code>	REAL	Bessel--
<code>r_j1( x )</code>	REAL	--
<code>r_jn(n, x )</code>	REAL	
<code>ir_finite( x )</code>	INTEGER	--
<code>ir_fp_class( x )</code>	INTEGER	--
<code>ir_ilogb( x )</code>	INTEGER	--
<code>ir_rint( x )</code>	INTEGER	--
<code>ir_isinf( x )</code>	INTEGER	--
<code>ir_isnan( x )</code>	INTEGER	--
<code>ir_isnormal( x )</code>	INTEGER	--
<code>ir_issubnormal( x )</code>	INTEGER	--
<code>ir_iszero( x )</code>	INTEGER	--
<code>ir_signbit( x )</code>	INTEGER	--
<code>r_addran()</code>	REAL	<b>r</b> andomnumbergenerators
<code>r_addrans( x, p, l, u )</code>	subroutineREAL	
<code>r_lcran()</code>	subroutine	
<code>r_lcrans( x, p, l, u )</code>	subroutine	
<code>r_shufrans(x, p, l, u)</code>		
<code>r_lgamma( x )</code>	REAL	<b>l</b> og <b>g</b> amma
<code>r_logb( x )</code>	REAL	--
<code>r_log1p( x )</code>	REAL	--
<code>r_log2( x )</code>	REAL	--

TABLE 1-2 Single-Precision Math Functions (Continued)

Function Name	Return Type	Description
<code>r_max_normal()</code>	REAL	
<code>r_max_subnormal()</code>	REAL	
<code>r_min_normal()</code>	REAL	
<code>r_min_subnormal()</code>	REAL	
<code>r_nextafter( x, y )</code>	REAL	
<code>r_quiet_nan( n )</code>	REAL	
<code>r_remainder( x, y )</code>	REAL	
<code>r_rint( x )</code>	REAL	
<code>r_scalb( x, y )</code>	REAL	
<code>r_scalbn( x, n )</code>	REAL	
<code>r_signaling_nan( n )</code>	REAL	
<code>r_significand( x )</code>	REAL	
<code>r_sin( x )</code>	REAL	sine
<code>r_sind( x )</code>	REAL	--
<code>r_sinh( x )</code>	REAL	hyperb sin
<code>r_sinp( x )</code>	REAL	--
<code>r_sinpi( x )</code>	REAL	--
<code>r_sincos( x, s, c )</code>	subroutine	sine & cosine
<code>r_sincosd( x, s, c )</code>	subroutine	--
<code>r_sincosp( x, s, c )</code>	subroutine	--
<code>r_sincospi( x, s, c )</code>	subroutine	--
<code>r_tan( x )</code>	REAL	tangent
<code>r_tand( x )</code>	REAL	--
<code>r_tanh( x )</code>	REAL	hyperb tan
<code>r_tanp( x )</code>	REAL	--
<code>r_tanpi( x )</code>	REAL	--
<code>r_y0( x )</code>	REAL	bessel
<code>r_y1( x )</code>	REAL	--
<code>r_yn( n, x )</code>	REAL	--

- Variables `c`, `l`, `p`, `s`, `u`, `x`, and `y` are of type **REAL**. Variable `n` is of type **INTEGER**.

- Type these functions as explicitly **REAL** if an **IMPLICIT** statement is in effect that types names starting with “r” to some other data type.
- **sind(x)**, **asind(x)**, ... take *degrees* rather than *radians*.

See also: **intro(3M)** and the *Numerical Computation Guide*.

## 1.3.2 Double-Precision Functions

The following subprograms are double-precision math functions and subroutines.

In general, these functions do *not* correspond to standard Fortran generic intrinsic functions—data types are determined by the usual data typing rules.

These **DOUBLE PRECISION** functions need to appear in a **DOUBLE PRECISION** statement.

Refer to the C library man pages for details: the man page for **d\_acos(x)** is **acos(3M)**

TABLE 1-3 Double Precision Math Functions

Function Name	Return Type	Description
<b>d_acos( x )</b>	<b>DOUBLE PRECISION</b>	arc cosine
<b>d_acosd( x )</b>	<b>DOUBLE PRECISION</b>	--
<b>d_acosh( x )</b>	<b>DOUBLE PRECISION</b>	arc cosh
<b>d_acosp( x )</b>	<b>DOUBLE PRECISION</b>	--
<b>d_acospi( x )</b>	<b>DOUBLE PRECISION</b>	--
<b>d_atan( x )</b>	<b>DOUBLE PRECISION</b>	arc tangent
<b>d_atand( x )</b>	<b>DOUBLE PRECISION</b>	--
<b>d_atanh( x )</b>	<b>DOUBLE PRECISION</b>	arc tanh
<b>d_atanp( x )</b>	<b>DOUBLE PRECISION</b>	--
<b>d_atanpi( x )</b>	<b>DOUBLE PRECISION</b>	--
<b>d_asin( x )</b>	<b>DOUBLE PRECISION</b>	arc sine
<b>d_asind( x )</b>	<b>DOUBLE PRECISION</b>	--
<b>d_asinh( x )</b>	<b>DOUBLE PRECISION</b>	arc sinh
<b>d_asinp( x )</b>	<b>DOUBLE PRECISION</b>	--
<b>d_asinpi( x )</b>	<b>DOUBLE PRECISION</b>	--

TABLE 1-3 Double Precision Math Functions (Continued)

Function Name	Return Type	Description
d_atan2( ( y, x )	DOUBLE PRECISION	arc tangent
d_atan2d( y, x )	DOUBLE PRECISION	--
d_atan2pi( y, x )	DOUBLE PRECISION	--
d_cbrt( x )	DOUBLE PRECISION	cube root
d_ceil( x )	DOUBLE PRECISION	ceiling
d_copysign( x, x )	DOUBLE PRECISION	--
d_cos( x )	DOUBLE PRECISION	cosine
d_cosd( x )	DOUBLE PRECISION	--
d_cosh( x )	DOUBLE PRECISION	hyperb cos
d_cosp( x )	DOUBLE PRECISION	--
d_cospi( x )	DOUBLE PRECISION	--
d_erf( x )	DOUBLE PRECISION	error func
d_erfc( x )	DOUBLE PRECISION	--
d_expm1( x )	DOUBLE PRECISION	(e**x)-1
d_floor( x )	DOUBLE PRECISION	floor
d_hypot( x, y )	DOUBLE PRECISION	hypotenuse
d_infinity( )	DOUBLE PRECISION	--
d_j0( x )	DOUBLE PRECISION	Bessel
d_j1( x )	DOUBLE PRECISION	--
d_jn( n, x )	DOUBLE PRECISION	--

TABLE 1-3 Double Precision Math Functions (Continued)

Function Name	Return Type	Description
<code>id_finite( x )</code>	INTEGER	
<code>id_fp_class( x )</code>	INTEGER	
<code>id_ilogb( x )</code>	INTEGER	
<code>id_rint( x )</code>	INTEGER	
<code>id_isinf( x )</code>	INTEGER	
<code>id_isnan( x )</code>	INTEGER	
<code>id_isnormal( x )</code>	INTEGER	
<code>id_issubnormal( x )</code>	INTEGER	
<code>id_iszero( x )</code>	INTEGER	
<code>id_signbit( x )</code>	INTEGER	
<code>d_addran()</code>	DOUBLE PRECISION	random number generators
<code>d_addrans(x, p, l, u)</code>	subroutine	
<code>d_lcran()</code>	DOUBLE PRECISION	
<code>d_lcrans(x, p, l, u)</code>	subroutine	
<code>d_shufrans(x, p, l,u)</code>	subroutine	
<code>d_lgamma( x )</code>	DOUBLE PRECISION	log gamma
<code>d_logb( x )</code>	DOUBLE PRECISION	--
<code>d_log1p( x )</code>	DOUBLE PRECISION	--
<code>d_log2( x )</code>	DOUBLE PRECISION	--

TABLE 1-3 Double Precision Math Functions (Continued)

Function Name	Return Type	Description
d_max_normal()	DOUBLE PRECISION	
d_max_subnormal()	DOUBLE PRECISION	
d_min_normal()	DOUBLE PRECISION	
d_min_subnormal()	DOUBLE PRECISION	
d_nextafter( x, y )	DOUBLE PRECISION	
d_quiet_nan( n )	DOUBLE PRECISION	
d_remainder( x, y )	DOUBLE PRECISION	
d_rint( x )	DOUBLE PRECISION	
d_scalb( x, y )	DOUBLE PRECISION	
d_scalbn( x, n )	DOUBLE PRECISION	
d_signaling_nan( n )	DOUBLE PRECISION	
d_significand( x )	DOUBLE PRECISION	
d_sin( x )	DOUBLE PRECISION	sine
d_sind( x )	DOUBLE PRECISION	--
d_sinh( x )	DOUBLE PRECISION	hyper sine
d_sinp( x )	DOUBLE PRECISION	--
d_sinpi( x )	DOUBLE PRECISION	--
d_sincos( x, s, c )	subroutine	sine & cosine
d_sincosd( x, s, c )	subroutine	--
d_sincosp( x, s, c )	subroutine	--
d_sincospi( x, s, c )	subroutine	
d_tan( x )	DOUBLE PRECISION	tangent
d_tand( x )	DOUBLE PRECISION	--
d_tanh( x )	DOUBLE PRECISION	hyperb tan
d_tanp( x )	DOUBLE PRECISION	--
d_tanpi( x )	DOUBLE PRECISION	--
d_y0( x )	DOUBLE PRECISION	bessel
d_y1( x )	DOUBLE PRECISION	--
d_yn( n, x )	DOUBLE PRECISION	--

- Variables **c**, **l**, **p**, **s**, **u**, **x**, and **y** are of type **DOUBLE PRECISION**. Variable **n** is of type **INTEGER**.

- Explicitly type these functions on a **DOUBLE PRECISION** statement or with an appropriate **IMPLICIT** statement).
- `sind(x)`, `asind(x)`, ... take *degrees* rather than *radians*.

See also: `intro(3M)` and the *Numerical Computation Guide*.

## 1.3.3 Quad-Precision Functions

These subprograms are **quadruple**-precision (**REAL\*16**) math functions and subroutines.

In general, these do *not* correspond to standard generic intrinsic functions; data types are determined by the usual data typing rules.

The quadruple precision functions must appear in a **REAL\*16** statement

TABLE 1-4 Quadruple-Precision **libm** Functions

Function Name	Return Type
<code>q_copysign( x, y )</code>	<b>REAL*16</b>
<code>q_fabs( x )</code>	<b>REAL*16</b>
<code>q_fmod( x )</code>	<b>REAL*16</b>
<code>q_infinity( )</code>	<b>REAL*16</b>
<code>iq_finite( x )</code>	<b>INTEGER</b>
<code>iq_fp_class( x )</code>	<b>INTEGER</b>
<code>iq_ilogb( x )</code>	<b>INTEGER</b>
<code>iq_isinf( x )</code>	<b>INTEGER</b>
<code>iq_isnan( x )</code>	<b>INTEGER</b>
<code>iq_isnormal( x )</code>	<b>INTEGER</b>
<code>iq_issubnormal( x )</code>	<b>INTEGER</b>
<code>iq_iszero( x )</code>	<b>INTEGER</b>
<code>iq_signbit( x )</code>	<b>INTEGER</b>

TABLE 1-4 Quadruple-Precision **libm** Functions (Continued)

Function Name	Return Type
<code>q_max_normal()</code>	<b>REAL*16</b>
<code>q_max_subnormal()</code>	<b>REAL*16</b>
<code>q_min_normal()</code>	<b>REAL*16</b>
<code>q_min_subnormal()</code>	<b>REAL*16</b>
<code>q_nextafter( x, y )</code>	<b>REAL*16</b>
<code>q_quiet_nan( n )</code>	<b>REAL*16</b>
<code>q_remainder( x, y )</code>	<b>REAL*16</b>
<code>q_scalbn( x, n )</code>	<b>REAL*16</b>
<code>q_signaling_nan( n )</code>	<b>REAL*16</b>

- The variables **c**, **l**, **p**, **s**, **u**, **x**, and **y** are of type quadruple precision. Variable **n** is of type **INTEGER**.
- Explicitly type these functions with a **REAL\*16** statement or with an appropriate **IMPLICIT** statement.
- **sind(x)**, **asind(x)**, ... take *degrees* rather than *radians*.

If you need to use any other quadruple-precision **libm** function, you can call it using **\$PRAGMA C(fcn)** before the call. For details, see the chapter on the C–Fortran interface in the *Fortran Programming Guide*.

## 1.4 Fortran Library Routines Reference

This section details the subroutines and functions in the Fortran library that are part of the Sun Studio Fortran 95 software but are not standard Fortran 95 intrinsics.

A synopsis of the calling interface is presented in a table of the form

<i>data declarations</i>			
<i>calling prototype synopsis with arguments</i>			
<i>argument 1 name</i>	<i>data type</i>	<i>input/output</i>	<i>description</i>
<i>argument 2 name</i>	<i>data type</i>	<i>input/output</i>	<i>description</i>
<i>Return value</i>	<i>data type</i>	Output	<i>description</i>

Additional man pages are available in section 3f of the Sun Studio man pages. For example, the command **man -s 3f** access will display the man page for the **access()** function. References to



man pages appear in this manual as *manpagename(section)*. For example, a reference to the man page for the **access()** function appears as **access(3f)**, and the man page for the Fortran 95 compiler as **f95(1)**

## 1.4.1 abort: Terminate and Write Core File

The subroutine is called by:

```
call abort
```

**abort** flushes the I/O buffers and then aborts the process, possibly producing a **core** file memory dump in the current directory. See **limit(1)** about limiting or suppressing core dumps.

## 1.4.2 access: Check File Permissions or Existence

The function is called by:

---

**INTEGER\*4 access**

**status = access ( name, mode )**

<i>name</i>	<b>character</b>	Input	File name
<i>mode</i>	<b>character</b>	Input	Permissions
Return value	<b>INTEGER*4</b>	Output	<i>status</i> =0: OK <i>status</i> >0: Error code

**access** determines if you can access the file *name* with the permissions specified by *mode*. **access** returns zero if the access specified by *mode* would be successful. See also **gerror(3F)** to interpret error codes.

Set *mode* to one or more of **r**, **w**, **x**, in any order or combination, or blank, where **r**, **w**, **x** have the following meanings:

'r'	Test for read permission
'w'	Test for write permission
'x'	Test for execute permission
' '	Test for existence of the file

Example 1: Test for read/write permission:

```

INTEGER*4 access, status
status = access ( 'taccess.data', 'rw' )
if ( status .eq. 0 ) write(*,*) "ok"
if ( status .ne. 0 ) write(*,*) 'cannot read/write', status

```

Example 2: Test for existence:

```

INTEGER*4 access, status
status = access ( 'taccess.data', ' ' ) ! blank mode
if ( status .eq. 0 ) write(*,*) "file exists"
if ( status .ne. 0 ) write(*,*) 'no such file', status

```

## 1.4.3 **alarm: Call Subroutine After a Specified Time**

The function is called by:

<b>INTEGER*4 alarm</b>			
<b>n = alarm ( time, sbrtn )</b>			
<i>time</i>	<b>INTEGER*4</b>	Input	Number of seconds to wait (0=do not call)
<i>sbrtn</i>	Routine name	Input	Subprogram to execute must be listed in an external statement.
Return value	<b>INTEGER*4</b>	Output	Time remaining on the last alarm

Example: **alarm**—wait 9 seconds then call **sbrtn**:

```

integer*4 alarm, time / 1 /
common / alarmcom / i
external sbrtn
i = 9
write(*,*) i
nseconds = alarm ( time, sbrtn )
do n = 1,100000 ! Wait until alarm activates sbrtn.
  r = n ! (any calculations that take enough time)
  x=sqrt(r)
end do
write(*,*) i
end
subroutine sbrtn
common / alarmcom / i
i = 3 ! Do no I/O in this routine.
return
end

```

See also: **alarm**(3C), **sleep**(3F), and **signal**(3F). Note the following restrictions:

- A subroutine cannot pass its own name to **alarm**.
- The **alarm** routine generates signals that could interfere with any I/O. The called subroutine, *sbrtn*, must not do any I/O itself.
- Calling **alarm()** from a parallelized or multi-threaded Fortran program may have unpredictable results.

## 1.4.4 bit: Bit Functions: and, or, ..., bit, setbit, ...

The definitions are:

<b>and</b> ( <i>word1</i> , <i>word2</i> )	Computes bitwise <i>and</i> of its arguments.
<b>or</b> ( <i>word1</i> , <i>word2</i> )	Computes bitwise <i>inclusive or</i> of its arguments.
<b>xor</b> ( <i>word1</i> , <i>word2</i> )	Computes bitwise <i>exclusive or</i> of its arguments.
<b>not</b> ( <i>word</i> )	Returns bitwise <i>complement</i> of its argument.
<b>lshift</b> ( <i>word</i> , <i>nbits</i> )	Logical left shift with no end around carry.
<b>rshift</b> ( <i>word</i> , <i>nbits</i> )	Arithmetic right shift with sign extension.
<b>call bis</b> ( <i>bitnum</i> , <i>word</i> )	Sets bit <i>bitnum</i> in <i>word</i> to 1.
<b>call bic</b> ( <i>bitnum</i> , <i>word</i> )	Clears bit <i>bitnum</i> in <i>word</i> to 0.
<b>bit</b> ( <i>bitnum</i> , <i>word</i> )	Tests bit <i>bitnum</i> in <i>word</i> and returns <b>LOGICAL .true.</b> if the bit is 1, <b>.false.</b> if it is 0.
<b>call setbit</b> ( <i>bitnum</i> , <i>word</i> , <i>state</i> )	Sets bit <i>bitnum</i> in <i>word</i> to 1 if <i>state</i> is nonzero, and clears it otherwise.

The alternate external versions for MIL-STD-1753 are:

<b>iand</b> ( <i>m</i> , <i>n</i> )	Computes the bitwise <i>and</i> of its arguments.
<b>ior</b> ( <i>m</i> , <i>n</i> )	Computes the bitwise <i>inclusive or</i> of its arguments.
<b>ieor</b> ( <i>m</i> , <i>n</i> )	Computes the bitwise <i>exclusive or</i> of its arguments.
<b>ishft</b> ( <i>m</i> , <i>k</i> )	Is a logical shift with no end around carry (left if <i>k</i> >0, right if <i>k</i> <0).
<b>ishftc</b> ( <i>m</i> , <i>k</i> , <i>ic</i> )	Circular shift: right-most <i>ic</i> bits of <i>m</i> are left-shifted circularly <i>k</i> places.
<b>ibits</b> ( <i>m</i> , <i>i</i> , <i>len</i> )	Extracts bits: from <i>m</i> , starting at bit <i>i</i> , extracts <i>len</i> bits.
<b>ibset</b> ( <i>m</i> , <i>i</i> )	Sets bit: return value is equal to word <i>m</i> with bit number <i>i</i> set to 1.

<code>ibclr( m, i )</code>	Clears bit: return value is equal to word <i>m</i> with bit number <i>i</i> set to 0.
<code>btest( m, i )</code>	Tests bit <i>i</i> in <i>m</i> ; returns <b>LOGICAL .true.</b> if the bit is 1, and <b>.false.</b> if it is 0.

See also “[1.4.36 mvbits: Move a Bit Field](#)” on page 66, and Chapters 2 and 3 for other functions that manipulate bit fields.

### 1.4.4.1 Usage: and, or, xor, not, rshift, lshift

For the intrinsic functions:

---

```
x = and( word1, word2 )
```

---

```
x = or( word1, word2 )
```

---

```
x = xor( word1, word2 )
```

---

```
x = not( word )
```

---

```
x = rshift( word, nbits )
```

---

```
x = lshift( word, nbits )
```

---

*word*, *word1*, *word2*, *nbits* are integer input arguments. These are intrinsic functions expanded inline by the compiler. The data type returned is that of the first argument.

No test is made for a reasonable value of *nbits*.

Example: **and, or, xor, not:**

```
demo% cat tandornot.f
      print 1, and(7,4), or(7,4), xor(7,4), not(4)
1     format(4x 'and(7,4)', 5x 'or(7,4)', 4x 'xor(7,4)',
1           6x 'not(4)'/4o12.11)
      end
demo% f95 tandornot.f
demo% a.out
      and(7,4)    or(7,4)    xor(7,4)    not(4)
000000000004 000000000007 000000000003 37777777773
demo%
```

Example: **lshift, rshift:**

```
demo% cat tlrshift.f
      integer*4 lshift, rshift
      print 1, lshift(7,1), rshift(4,1)
1     format(1x 'lshift(7,1)', 1x 'rshift(4,1)'/2o12.11)
```

```

        end
demo% f95 tlrshift.f
demo% a.out
      lshift(7,1) rshift(4,1)
      00000000016 00000000002
demo%

```

## 1.4.4.2 Usage: **bic**, **bis**, **bit**, **setbit**

For the subroutines and functions

---

```
call bic( bitnum, word )
```

---

```
call bis( bitnum, word )
```

---

```
call setbit( bitnum, word, state )
```

---

**LOGICAL** *bit*

```
x = bit( bitnum, word )
```

---

*bitnum*, *state*, and *word* are **INTEGER\*4** input arguments. Function **bit**() returns a logical value.

Bits are numbered so that bit 0 is the least significant bit, and bit 31 is the most significant.

**bic**, **bis**, and **setbit** are external subroutines. **bit** is an external function.

Example 3: **bic**, **bis**, **setbit**, **bit**:

```

integer*4 bitnum/2/, state/0/, word/7/
logical bit
print 1, word
1  format(13x 'word', o12.11)
   call bic( bitnum, word )
   print 2, word
2  format('after bic(2,word)', o12.11)
   call bis( bitnum, word )
   print 3, word
3  format('after bis(2,word)', o12.11)
   call setbit( bitnum, word, state )
   print 4, word
4  format('after setbit(2,word,0)', o12.11)
   print 5, bit(bitnum, word)
5  format('bit(2,word)', L )
   end
< output>
      word 00000000007
after bic(2,word) 00000000003

```

```

after bis(2,word) 00000000007
after setbit(2,word,0) 00000000003
bit(2,word) F

```

## 1.4.5 chdir: Change Default Directory

The function is called by:

---

**INTEGER\*4** *chdir*

*n* = **chdir**( *dirname* )

<i>dirname</i>	<b>character</b>	Input	Directory name
Return value	<b>INTEGER*4</b>	Output	<i>n</i> =0: OK, <i>n</i> >0: Error code

Example: **chdir**—change **cwd** to **MyDir**:

```

INTEGER*4 chdir, n
n = chdir ( 'MyDir' )
if ( n.ne. 0 ) stop 'chdir: error'
end

```

See also: **chdir**(2), **cd**(1), and **gerror**(3F) to interpret error codes.

Path names can be no longer than **MAXPATHLEN** as defined in **<sys/param.h>**. They can be relative or absolute paths.

Use of this function can cause inquire by unit to fail.

Certain Fortran file operations reopen files by name. Using **chdir** while doing I/O can cause the runtime system to lose track of files created with relative path names, including the files that are created by open statements without file names.

## 1.4.6 chmod: Change the Mode of a File

The function is called by:

---

**INTEGER\*4** *chmod*

*n* = **chmod**( *name*, *mode* )

<i>name</i>	<b>character</b>	Input	Path name
-------------	------------------	-------	-----------

<i>mode</i>	<b>character</b>	Input	Anything recognized by <i>chmod</i> (1), such as <b>o-w, 444</b> , etc.
Return value	<b>INTEGER*4</b>	Output	<i>n</i> = 0: OK; <i>n</i> >0: System error number

Example: **chmod**—add write permissions to **MyFile**:

```

character*18 name, mode
INTEGER*4 chmod, n
name = 'MyFile'
mode = '+w'
n = chmod( name, mode )
if ( n .ne. 0 ) stop 'chmod: error'
end

```

See also: **chmod**(1), and **gerror**(3F) to interpret error codes.

Path names cannot be longer than **MAXPATHLEN** as defined in `<sys/param.h>`. They can be relative or absolute paths.

## 1.4.7 date: Get Current Date as a Character String

**Note** – This routine is not “Year 2000 Safe” because it returns only a two-digit value for the year. Programs that compute differences between dates using the output of this routine may not work properly after 31 December, 1999. Programs using this **date**( ) routine will see a runtime warning message the first time the routine is called to alert the user. See **date\_and\_time**( ) as a possible alternate routine.

The subroutine is called by:

```
call date( c )
```

<i>c</i>	<b>CHARACTER*9</b>	Output	Variable, array, array element, or character substring
----------	--------------------	--------	--

The form of the returned string *c* is *dd-mmm-yy*, where *dd* is the day of the month as a 2-digit number, *mmm* is the month as a 3-letter abbreviation, and *yy* is the year as a 2-digit number (and is not year 2000 safe!).

Example: **date**:

```

demo% cat dat1.f
* dat1.f -- Get the date as a character string.
character c*9

```

```

        call date ( c )
        write(*,"(' The date today is: ', A9 )" ) c
    end
demo% f95 dat1.f
demo% a.out
Computing time differences using the 2 digit year from subroutine
        date is not safe after year 2000.
    The date today is: 9-Jan-02
demo%

```

See also `idate()` and `date_and_time()`.

### 1.4.7.1 `date_and_time`: Get Date and Time

This is a Fortran 95 intrinsic routine, and is Year 2000 safe.

The `date_and_time` subroutine returns data from the real-time clock and the date. Local time is returned, as well as the difference between local time and Universal Coordinated Time (UTC) (also known as Greenwich Mean Time, GMT).

The `date_and_time()` subroutine is called by:

---

```
call date_and_time( date, time, zone, values )
```

---

<i>date</i>	<b>CHARACTER*8</b>	Output	Date, in form CCYYMMDD, where CCYY is the four-digit year, MM the two-digit month, and DD the two-digit day of the month. For example: 19980709
<i>time</i>	<b>CHARACTER*10</b>	Output	The current time, in the form hhmmss.sss, where hh is the hour, mm minutes, and ss.sss seconds and milliseconds.
<i>zone</i>	<b>CHARACTER*5</b>	Output	The time difference with respect to UTC, expressed in hours and minutes, in the form hhmm
<i>values</i>	<b>INTEGER*4 VALUES (8)</b>	Output	An integer array of 8 elements described below.

The eight values returned in the **INTEGER\*4** *values* array are

<b>VALUES (1)</b>	The year, as a 4-digit integer. For example, 1998.
<b>VALUES (2)</b>	The month, as an integer from 1 to 12.
<b>VALUES (3)</b>	The day of the month, as an integer from 1 to 31.



<b>VALUES (4)</b>	The time difference, in minutes, with respect to UTC.
<b>VALUES (5)</b>	The hour of the day, as an integer from 1 to 23.
<b>VALUES (6)</b>	The minutes of the hour, as an integer from 1 to 59.
<b>VALUES (7)</b>	The seconds of the minute, as an integer from 0 to 60.
<b>VALUES (8)</b>	The milliseconds of the second, in range 0 to 999.

An example using **date\_and\_time**:

```
demo% cat dtm.f
integer date_time(8)
character*10 b(3)
call date_and_time(b(1), b(2), b(3), date_time)
print *, 'date_time array values:'
print *, 'year=', date_time(1)
print *, 'month_of_year=', date_time(2)
print *, 'day_of_month=', date_time(3)
print *, 'time difference in minutes=', date_time(4)
print *, 'hour of day=', date_time(5)
print *, 'minutes of hour=', date_time(6)
print *, 'seconds of minute=', date_time(7)
print *, 'milliseconds of second=', date_time(8)
print *, 'DATE=', b(1)
print *, 'TIME=', b(2)
print *, 'ZONE=', b(3)
end
```

When run on a computer in California, USA on February 16, 2000, it generated the following output:

```
date_time array values:
year= 2000
month_of_year= 2
day_of_month= 16
time difference in minutes= -420
hour of day= 11
minutes of hour= 49
seconds of minute= 29
milliseconds of second= 236
DATE=20000216
TIME=114929.236
ZONE=-0700
```

## 1.4.8 **mtime, etime: Elapsed Execution Time**

Both functions have return values of elapsed time (or -1.0 as error indicator). The time returned is in seconds.

Versions of **mtime** and **etime** used by Fortran 95 use the system's low resolution clock by default. The resolution is one hundredth of a second. However, if the program is run under the Sun OS™ operating system utility **ptime(1)**, (`/usr/proc/bin/ptime`), the high resolution clock is used.

### 1.4.8.1 **mtime: Elapsed Time Since the Last mtime Call**

For **mtime**, the elapsed time is:

- First call: elapsed time since start of execution
- Subsequent calls: elapsed time since the last call to **mtime**
- Single processor: time used by the CPU
- Multiple Processor: the sum of times for all the CPUs, which is not useful data; use **etime** instead.

---

**Note** – Calling **mtime** from within a parallelized loop gives non-deterministic results, since the elapsed time counter is global to all threads participating in the loop

---

The function is called by:

<i>e</i> = <b>mtime</b> ( <i>tarray</i> )			
<i>tarray</i>	<b>real(2)</b>	Output	<i>e</i> = -1.0: Error: <i>tarray</i> values are undefined <i>e</i> ≠ -1.0: User time in <i>tarray</i> (1) if no error. System time in <i>tarray</i> (2) if no error
Return value	<b>real</b>	Output	<i>e</i> = -1.0: Error <i>e</i> ≠ -1.0: The sum of <i>tarray</i> (1) and <i>tarray</i> (2)

Example: **mtime()**, single processor:

```
demo% cat tdmtime.f
      real e, dtime, t(2)
      print *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
      do i = 1, 10000
         k=k+1
      end do
      e = dtime( t )
```

```

        print *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
    end
demo% f95 tdttime.f
demo% a.out
elapsed: 0.0E+0 , user: 0.0E+0 , sys: 0.0E+0
    elapsed: 0.03 , user: 0.01 , sys: 0.02
demo%
```

## 1.4.8.2 **etime**: Elapsed Time Since Start of Execution

For **etime**, the elapsed time is:

- Single Processor Execution—CPU time for the calling process
- Multiple Processor Execution—wallclock time while processing your program

The runtime library determines that a program is executing in a multiprocessor mode if either the **PARALLEL** or **OMP\_NUM\_THREADS** environment variables are defined to some integer value greater than 1.

The function is called by:

---

```
e = etime( tarray )
```

<i>tarray</i>	<b>real(2)</b>	Output	<i>e</i> = -1.0: Error: <i>tarray</i> values are undefined. <i>e</i> ≠ -1.0: Single Processor: User time in <i>tarray</i> (1). System time in <i>tarray</i> (2) Multiple Processor: Wall clock time in <i>tarray</i> (1), 0.0 in <i>tarray</i> (2)
Return value	<b>real</b>	Output	<i>e</i> = -1.0: Error <i>e</i> ≠ -1.0: The sum of <i>tarray</i> (1) and <i>tarray</i> (2)

Take note that the initial call to **etime** will be inaccurate. It merely enables the system clock. Do not use the value returned by the initial call to **etime**.

Example: **etime()**, single processor:

```

demo% cat tetime.f
    real e, etime, t(2)
    e = etime(t)          ! Startup etime - do not use result
    do i = 1, 10000
        k=k+1
    end do
    e = etime( t )
    print *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
end
```

```
demo% f95 tetime.f
demo% a.out
elapsed: 0.02 , user: 0.01 , sys: 0.01
demo%
```

See also `times(2)`, and the *Fortran Programming Guide*.

## 1.4.9 `exit`: Terminate a Process and Set the Status

The subroutine is called by:

---

<code>call exit( status )</code>		
<i>status</i>	INTEGER*4	Input

---

Example: `exit()`:

```
...
  if(dx .lt. 0.) call exit( 0 )
...
end
```

`exit` flushes and closes all the files in the process, and notifies the parent process if it is executing a `wait`.

The low-order 8 bits of *status* are available to the parent process. These 8 bits are shifted left 8 bits, and all other bits are zero. (Therefore, *status* should be in the range of 256- 65280). This call will never return.

The C function `exit` can cause cleanup actions before the final system `'exit'`.

Calling `exit` without an argument causes a compile-time warning message, and a zero will be automatically provided as an argument. See also: `exit(2)`, `fork(2)`, `fork(3F)`, `wait(2)`, `wait(3F)`.

## 1.4.10 `fdate`: Return Date and Time in an ASCII String

The subroutine or function is called by:

---

<code>call fdate( string )</code>		
<i>string</i>	character*24	Output

---

or:

<b>CHARACTER fdate*24</b>			If used as a function, the calling routine must define the type and size of <b>fdate</b> .
<i>string</i> = <b>fdate</b> ( )			
Return value	<b>character*24</b>	Output	

Example 1: **fdate** as a subroutine:

```

character*24 string
call fdate( string )
write(*,*) string
end

```

Output:

```
Wed Aug  3 15:30:23 1994
```

Example 2: **fdate** as a function, same output:

```

character*24 fdate
write(*,*) fdate()
end

```

See also: **ctime**(3), **time**(3F), and **idate**(3F).

## 1.4.11 **flush**: Flush Output to a Logical Unit

The function is called by:

<b>INTEGER*4 flush</b>			
<i>n</i> = <b>flush</b> ( <i>lunit</i> )			
<i>lunit</i>	<b>INTEGER*4</b>	Input	Logical unit
Return value	<b>INTEGER*4</b>	Output	<i>n</i> = 0 no error <i>n</i> > 0 error number

The **flush** function flushes the contents of the buffer for the logical unit, **lunit**, to the associated file. This is most useful for logical units 0 and 6 when they are both associated with the console terminal. The function returns a positive error number if an error was encountered; zero otherwise.

See also **fclose**(3S).

## 1.4.12 **fork: Create a Copy of the Current Process**

The function is called by:

---

**INTEGER\*4 fork**

*n* = **fork**()

---

Return value	<b>INTEGER*4</b>	Output	<i>n</i> >0: <i>n</i> =Process ID of copy <i>n</i> <0, <i>n</i> =System error code
--------------	------------------	--------	---

---

The **fork** function creates a copy of the calling process. The only distinction between the two processes is that the value returned to one of them, referred to as the *parent* process, will be the process ID of the copy. The copy is usually referred to as the *child* process. The value returned to the child process will be zero.

All logical units open for writing are flushed before the fork to avoid duplication of the contents of I/O buffers in the external files.

Example: **fork**():

```

INTEGER*4 fork, pid
pid = fork()
if(pid.lt.0) stop 'fork error'
if(pid.gt.0) then
    print *, 'I am the parent'
else
    print *, 'I am the child'
endif

```

A corresponding **exec** routine has not been provided because there is no satisfactory way to retain open logical units across the **exec** routine. However, the usual function of **fork/exec** can be performed using **system**(3F). See also: **fork**(2), **wait**(3F), **kill**(3F), **system**(3F), and **perror**(3F).

## 1.4.13 **fseek, ftell: Determine Position and Reposition a File**

**fseek** and **ftell** are routines that permit repositioning of a file. **ftell** returns a file's current position as an offset of so many bytes from the beginning of the file. At some later point in the program, **fseek** can use this saved offset value to reposition the file to that same place for reading.

### 1.4.13.1 **fseek: Reposition a File on a Logical Unit**

The function is called by:

---

**INTEGER\*4** `fseek`

`n = fseek( lunit, offset, from )`

<i>lunit</i>	<b>INTEGER*4</b>	Input	Open logical unit
<i>offset</i>	<b>INTEGER*4</b> <i>or</i> <b>INTEGER*8</b>	Input	Offset in bytes relative to position specified by <i>from</i>
	An <b>INTEGER*8</b> offset value is required when compiled for a 64-bit environment. If a literal constant is supplied, it must be a 64-bit constant, for example: <b>100_8</b>		
<i>from</i>	<b>INTEGER*4</b>	Input	0=Beginning of file 1=Current position 2=End of file
Return value	<b>INTEGER*4</b>	Output	<i>n</i> =0: OK; <i>n</i> >0: System error code

---

**Note** – On sequential files, following a call to **fseek** by an output operation (for example, **WRITE**) causes all data records following the **fseek** position to be deleted and replaced by the new data record (and an end-of-file mark). Rewriting a record in place can only be done with direct access files.

---

Example: **fseek()**—Reposition **MyFile** to two bytes from the beginning

```
INTEGER*4 fseek, lunit/1/, offset/2/, from/0/, n
open( UNIT=lunit, FILE='MyFile' )
n = fseek( lunit, offset, from )
if ( n .gt. 0 ) stop 'fseek error'
end
```

Example: Same example in a 64-bit environment and compiled with **-m64**:

```
INTEGER*4 fseek, lunit/1/, from/0/, n
INTEGER*8 offset/2/
open( UNIT=lunit, FILE='MyFile' )
n = fseek( lunit, offset, from )
if ( n .gt. 0 ) stop 'fseek error'
end
```

### 1.4.13.2 **ftell: Return Current Position of File**

The function is called by:

---

**INTEGER\*4 ftell**

*n* = **ftell**( *lunit* )

<i>lunit</i>	<b>INTEGER*4</b>	Input	Open logical unit
Return value	<b>INTEGER*4</b>	Output	<i>n</i> ≥ 0: <i>n</i> = Offset in bytes from start of file <i>n</i> < 0: <i>n</i> = System error code

---

Example: **ftell()**:

```

INTEGER*4 ftell, lunit/1/, n
open( UNIT=lunit, FILE='MyFile' )
...
n = ftell( lunit )
if ( n .lt. 0 ) stop 'ftell error'
...

```

Example: Same example in a 64-bit environment and compiled with **-m64**:

```

INTEGER*4 lunit/1/
INTEGER*8 ftell, n
open( UNIT=lunit, FILE='MyFile' )
...
n = ftell( lunit )
if ( n .lt. 0 ) stop 'ftell error'
...

```

See also **fseek(3S)** and **perror(3F)**; also **fseeko64(3F)** **ftello64(3F)**.

## 1.4.14 **fseeko64, ftello64: Determine Position and Reposition a Large File**

**fseeko64** and **ftello64** are "large file" versions of **fseek** and **ftell**. They take and return **INTEGER\*8** file position offsets. (A "large file" is larger than 2 Gigabytes and therefore a byte-position must be represented by a 64-bit integer.) Use these versions to determine and/or reposition large files.

### 1.4.14.1 **fseeko64: Reposition a File on a Logical Unit**

The function is called by:



**INTEGER fseeko64**

$n = \text{fseeko64}( \text{lunit}, \text{offset64}, \text{from} )$

<i>lunit</i>	<b>INTEGER*4</b>	Input	Open logical unit
<i>offset64</i>	<b>INTEGER*8</b>	Input	64-bit offset in bytes relative to position specified by <i>from</i>
<i>from</i>	<b>INTEGER*4</b>	Input	0=Beginning of file 1=Current position 2=End of file
Return value	<b>INTEGER*4</b>	Output	$n=0$ : OK; $n>0$ : System error code

**Note** – On sequential files, following a call to **fseeko64** by an output operation (for example, **WRITE**) causes all data records following the **fseek** position to be deleted and replaced by the new data record (and an end-of-file mark). Rewriting a record in place can only be done with direct access files.

Example: **fseeko64()** — Reposition **MyFile** to two bytes from the beginning:

```

INTEGER fseeko64, lunit/1/, from/0/, n
INTEGER*8 offset/200/
open( UNIT=lunit, FILE='MyFile' )
n = fseeko64( lunit, offset, from )
if ( n .gt. 0 ) stop 'fseek error'
end

```

**1.4.14.2****ftello64: Return Current Position of File**

The function is called by:

**INTEGER\*8 ftello64**

$n = \text{ftello64}( \text{lunit} )$

<i>lunit</i>	<b>INTEGER*4</b>	Input	Open logical unit
Return value	<b>INTEGER*8</b>	Output	$n \geq 0$ : $n$ =Offset in bytes from start of file $n < 0$ : $n$ =System error code

Example: **ftello64()**:

```

INTEGER*8 ftello64, lunit/1/, n
open( UNIT=lunit, FILE='MyFile' )
...
n = ftello64( lunit )
if ( n .lt. 0 ) stop 'ftell error'
...

```

## 1.4.15 getarg, iargc: Get Command-Line Arguments

**getarg** and **iargc** access arguments on the command line (after expansion by the command-line preprocessor).

### 1.4.15.1 getarg: Get a Command-Line Argument

The subroutine is called by:

call <b>getarg</b> ( <i>k</i> , <i>arg</i> )			
<i>k</i>	INTEGER*4	Input	Index of argument (0=first=command name)
<i>arg</i>	character*n	Output	<i>k</i> th argument
<i>n</i>	INTEGER*4	Size of <i>arg</i>	Large enough to hold longest argument

### 1.4.15.2 iargc: Get the Number of Command-Line Arguments

The function is called by:

<i>m</i> = <b>iargc</b> ( )			
Return value	INTEGER*4	Output	Number of arguments on the command line

Example: **iargc** and **getarg**, get argument count and each argument:

```

demo% cat yarg.f
      character argv*10
      INTEGER*4 i, iargc, n
      n = iargc( )
      do 1 i = 1, n
        call getarg( i, argv )
1      write( *, '( i2, 1x, a )' ) i, argv
      end
demo% f95 yarg.f
demo% a.out *.f
1 first.f
2 yarg.f

```

See also `execve(2)` and `getenv(3F)`.

## 1.4.16 `getc`, `fgetc`: Get Next Character

`getc` and `fgetc` get the next character from the input stream. Do not mix calls to these routines with normal Fortran I/O on the same logical unit.

### 1.4.16.1 `getc`: Get Next Character From `stdin`

The function is called by:

---

```
INTEGER*4 getc
```

```
status = getc( char )
```

---

<i>char</i>	<b>character</b>	Output	Next character
Return value	<b>INTEGER*4</b>	Output	<i>status</i> =0: OK <i>status</i> =-1: End of file <i>status</i> >0: System error code or <b>f95</b> I/O error code

---

Example: `getc` gets each character from the keyboard; note the Control-D (^D):

```
character char
INTEGER*4 getc, status
status = 0
do while ( status .eq. 0 )
  status = getc( char )
  write(*, '(i3, o4.3)') status, char
end do
end
```

After compiling, a sample run of the above source is:

```
demo% a.out
ab   Program reads letters typed in
0 141   Program outputs status and octal value of the characters entered
0 142   141 represents 'a', 142 is 'b'
0 012   012 represents the RETURN key
^D     terminated by a CONTROL-D.
-1 377   Next attempt to read returns CONTROL-D
demo%
```

For any logical unit, do not mix normal Fortran input with `getc()`.

## 1.4.16.2 **fgetc**: Get Next Character From Specified Logical Unit

The function is called by:

---

**INTEGER\*4 fgetc**

*status* = **fgetc**( *lunit*, *char* )

<i>lunit</i>	<b>INTEGER*4</b>	Input	Logical unit
<i>char</i>	<b>character</b>	Output	Next character
Return value	<b>INTEGER*4</b>	Output	<i>status</i> = -1: End of File <i>status</i> > 0: System error code or <b>f95</b> I/O error code

---

Example: **fgetc** gets each character from **tfgetc.data**; note the linefeeds (Octal 012):

```

character char
INTEGER*4 fgetc, status
open( unit=1, file='tfgetc.data' )
status = 0
do while ( status .eq. 0 )
    status = fgetc( 1, char )
    write(*, '(i3, o4.3)') status, char
end do
end

```

After compiling, a sample run of the above source is:

```

demo% cat tfgetc.data
ab
yz
demo% a.out
0 141    "a' read
0 142    "b' read
0 012    linefeed read
0 171    "y' read
0 172    "z' read
0 012    linefeed read
-1 012    CONTROL-D read
demo%

```

For any logical unit, do not mix normal Fortran input with **fgetc**( ).

See also: **getc**(3S), **intro**(2), and **perror**(3F).

## 1.4.17 **getcwd: Get Path of Current Working Directory**

The function is called by:

---

**INTEGER\*4** `getcwd`

`status = getcwd( dirname )`

<i>dirname</i>	<b>character*n</b>	Output The path of the current directory is returned	Path name of the current working directory. <i>n</i> must be large enough for longest path name
Return value	<b>INTEGER*4</b>	Output	<i>status</i> =0: OK <i>status</i> >0: Error code

Example: **getcwd**:

```

INTEGER*4 getcwd, status
character*64 dirname
status = getcwd( dirname )
if ( status .ne. 0 ) stop 'getcwd: error'
write(*,*) dirname
end

```

See also: **chdir**(3F), **perror**(3F), and **getwd**(3).

Note: the path names cannot be longer than **MAXPATHLEN** as defined in `<sys/param.h>`.

## 1.4.18 **getenv: Get Value of Environment Variables**

The subroutine is called by:

---

**call** `getenv( ename, value )`

<i>ename</i>	<b>character*n</b>	Input	Name of the environment variable sought
<i>value</i>	<b>character*n</b>	Output	Value of the environment variable found; blanks if not successful

The size of *ename* and *value* must be large enough to hold their respective character strings.

If *value* is too short to hold the complete string value, the string is truncated to fit in *value*.

The **getenv** subroutine searches the environment list for a string of the form *ename*=*value* and returns the value in *value* if such a string is present; otherwise, it fills *value* with blanks.

Example: Use `getenv()` to print the value of `$SHELL`:

```
character*18  evaluate
call getenv( 'SHELL', evaluate )
write(*,*) "'", evaluate, "'"
end
```

See also: `execve(2)` and `environ(5)`.

## 1.4.19 `getfd`: Get File Descriptor for External Unit Number

The function is called by:

---

**INTEGER\*4** `getfd`

*fildes* = `getfd( unitn )`

<i>unitn</i>	<b>INTEGER*4</b>	Input	External unit number
Return value	<b>INTEGER*4</b> or <b>INTEGER*8</b>	Output	File descriptor if file is connected; -1 if file is not connected An <b>INTEGER*8</b> result is returned when compiling for 64-bit environments

Example: `getfd()`:

```
INTEGER*4 fildes, getfd, unitn/1/
open( unitn, file='tgetfd.data' )
fildes = getfd( unitn )
if ( fildes .eq. -1 ) stop 'getfd: file not connected'
write(*,*) 'file descriptor = ', fildes
end
```

See also `open(2)`.

## 1.4.20 `getfilep`: Get File Pointer for External Unit Number

The function is:

---

*irtn* = `c_read( getfilep( unitn ), inbyte, 1 )`

<i>c_read</i>	C function	Input	User's own C function. See example.
<i>unitn</i>	<b>INTEGER*4</b>	Input	External unit number.

<b>getfilep</b>	<b>INTEGER*4</b> or <b>INTEGER*8</b>	Return value	File pointer if the file is connected; -1 if the file is not connected. An <b>INTEGER*8</b> value is returned when compiling for 64-bit environments
-----------------	---	--------------	--

This function is used for mixing standard Fortran I/O with C I/O. Such a mix is nonportable, and is not guaranteed for subsequent releases of the operating system or Fortran. Use of this function is not recommended, and no direct interface is provided. You must create your own C routine to use the value returned by **getfilep**. A sample C routine is shown below.

Example: Fortran uses **getfilep** by passing it to a C function:

```
demo% cat tgetfilepF.f
```

```

character*1 inbyte
integer*4   c_read, getfilep, unitn / 5 /
external   getfilep
write(*,'(a,$)') 'What is the digit? '

irtn = c_read( getfilep( unitn ), inbyte, 1 )

write(*,9) inbyte
9   format('The digit read by C is ', a )
end
```

Sample C function actually using **getfilep**:

```
demo% cat tgetfilepC.c
```

```

#include <stdio.h>
int c_read_ ( fd, buf, nbytes, buf_len )
FILE **fd ;
char *buf ;
int *nbytes, buf_len ;
{
    return fread( buf, 1, *nbytes, *fd ) ;
}
```

A sample compile-build-run is:

```

demo% cc -c tgetfilepC.c
demo% f95 tgetfilepC.o tgetfilepF.f
demo% a.out
What is the digit? 3
The digit read by C is 3
demo%
```

For more information, read the chapter on the C-Fortran interface in the *Fortran Programming Guide*. See also **open(2)**.

## 1.4.21 **getlog: Get User's Login Name**

The subroutine is called by:

---

**call getlog( name )**

<i>name</i>	<b>character*n</b>	Output	User's login name, or all blanks if the process is running detached from a terminal. <i>n</i> should be large enough to hold the longest name.

---

Example: **getlog**:

```

character*18 name
call getlog( name )
write(*,*) "'", name, "'"
end

```

See also **getlogin(3)**.

## 1.4.22 **getpid: Get Process ID**

The function is called by:

---

**INTEGER\*4 getpid**

*pid* = **getpid()**

Return value	<b>INTEGER*4</b>	Output	Process ID of the current process

---

Example: **getpid**:

```

INTEGER*4 getpid, pid
pid = getpid()
write(*,*) 'process id = ', pid
end

```

See also **getpid(2)**.

## 1.4.23 **getuid, getgid: Get User or Group ID of Process**

**getuid** and **getgid** get the user or group ID of the process, respectively.

### 1.4.23.1 **getuid: Get User ID of the Process**

The function is called by:



**INTEGER\*4 getuid***uid* = **getuid()**

Return value	<b>INTEGER*4</b>	Output	User ID of the process
--------------	------------------	--------	------------------------

**1.4.23.2 getgid: Get Group ID of the Process**

The function is called by:

**INTEGER\*4 getgid***gid* = **getgid()**

Return value	<b>INTEGER*4</b>	Output	Group ID of the process
--------------	------------------	--------	-------------------------

Example: **getuid()** and **getpid()**:

```

INTEGER*4 getuid, getgid, gid, uid
uid = getuid()
gid = getgid()
write(*,*) uid, gid
end

```

See also: **getuid(2)**.**1.4.24 hostnm: Get Name of Current Host**

The function is called by:

**INTEGER\*4 hostnm***status* = **hostnm( name )**

<i>name</i>	<b>character*n</b>	Output	Name of current host system. <i>n</i> must be large enough to hold the host name.
Return value	<b>INTEGER*4</b>	Output	<i>status</i> =0: OK <i>status</i> >0: Error

Example: **hostnm()**:

```

INTEGER*4 hostnm, status
character*8 name
status = hostnm( name )

```

```

write(*,*) 'host name = "', name, '"'
end

```

See also `gethostname(2)`.

## 1.4.25 `idate`: Return Current Date

`idate` puts the current system date into one integer array: day, month, and year.

The subroutine is called by:

---

**call `idate`( *iarray* )** *Standard Version*

<i>iarray</i>	<b>INTEGER*4</b>	Output	Three-element array: day, month, year.
---------------	------------------	--------	--

Example: `idate` (standard version):

```

demo% cat tidate.f
      INTEGER*4 iarray(3)
      call idate( iarray )
      write(*, "( ' The date is: ',3i5)" ) iarray
end
demo% f95 tidate.f
demo% a.out
The date is: 10 8 1998
demo%

```

## 1.4.26 `ieee_flags`, `ieee_handler`, `sigfpe`: IEEE Arithmetic

These subprograms provide modes and status required to fully exploit ANSI/IEEE Standard 754-1985 arithmetic in a Fortran program. They correspond closely to the functions `ieee_flags(3M)`, `ieee_handler(3M)`, and `sigfpe(3)`.

Here is a summary:

TABLE 1-5 IEEE Arithmetic Support Routines

<b>ieeeer = ieee_flags( action, mode, in, out )</b>		
<b>ieeeer = ieee_handler( action, exception, hdl )</b>		
<b>ieeeer = sigfpe( code, hdl )</b>		
<i>action</i>	<b>character</b>	Input
<i>code</i>	<b>sigfpe_code_type</b>	Input
<i>mode</i>	<b>character</b>	Input
<i>in</i>	<b>character</b>	Input
<i>exception</i>	<b>character</b>	Input
<i>hdl</i>	<b>sigfpe_handler_type</b>	Input
<i>out</i>	<b>character</b>	Output
Return value	<b>INTEGER*4</b>	Output

See the *Numerical Computation Guide* for details on how these functions can be used strategically.

If you use **sigfpe**, you must do your own setting of the corresponding trap-enable-mask bits in the floating-point status register. The details are in the SPARC architecture manual. The **libm** function **ieee\_handler** sets these trap-enable-mask bits for you.

The character keywords accepted for *mode* and *exception* depend on the value of *action*.

TABLE 1-6 **ieee\_flags( action, mode, in, out )** Parameters and Actions

<i>action</i> = 'clearall'	<i>mode, in, out</i> , unused; returns 0	
<i>action</i> = 'clear'	<i>mode</i> = 'direction'	
	<i>mode</i> = 'exception'	<i>in</i> = 'inexact' or 'division' or 'underflow' or 'overflow' or 'invalid' or 'all' or 'common'
clear <i>mode, in</i> <i>out</i> is unused; returns 0		

TABLE 1-6 `ieee_flags`(*action*, *mode*, *in*, *out*) Parameters and Actions (Continued)

<p><i>action</i> = 'set'</p> <p>set floating-point <i>mode</i>, <i>in</i></p> <p><i>out</i> is unused; returns 0</p>	<p><i>mode</i> = 'direction'</p>	<p><i>in</i> = 'nearest' or</p> <p>'tozero' or</p> <p>'positive' or</p> <p>'negative'</p>
	<p><i>mode</i> = 'exception'</p>	<p><i>in</i> = 'inexact' or</p> <p>'division' or</p> <p>'underflow' or</p> <p>'overflow' or</p> <p>'invalid' or</p> <p>'all' or</p> <p>'common'</p>
<p><i>action</i> = 'get'</p> <p>test <i>mode</i> settings</p> <p><i>in</i>, <i>out</i> may be blank or one of the settings to test returns the current setting depending on <i>mode</i>, or 'not available' The function returns 0 or the current exception flags if <i>mode</i> = 'exception'</p>	<p><i>mode</i> = 'direction'</p>	<p><i>out</i> = 'nearest' or</p> <p>'tozero' or</p> <p>'positive' or</p> <p>'negative'</p>
	<p><i>mode</i> = 'exception'</p>	<p><i>out</i> = 'inexact' or</p> <p>'division' or</p> <p>'underflow' or</p> <p>'overflow' or</p> <p>'invalid' or</p> <p>'all' or</p> <p>'common'</p>

TABLE 1-7 `ieee_handler`(*action*, *in*, *out*) Parameters

<p><i>action</i> = 'clear'</p> <p>clear user exception handling of <i>in</i>; <i>out</i> is unused</p>	<p><i>in</i> = 'inexact' or</p> <p>'division' or</p> <p>'underflow' or</p> <p>'overflow' or</p> <p>'invalid' or</p> <p>'all' or</p> <p>'common'</p>
--	---

TABLE 1-7 `ieee_handler( action, in, out)` Parameters (Continued)

<code>action = 'set'</code>	<code>in = 'inexact' or</code>
set user exception handling of <i>in</i> ; <i>out</i> is address of handler routine, or <code>SIGFPE_DEFAULT</code> , or <code>SIGFPE_ABORT</code> , or <code>SIGFPE_IGNORE</code> defined in	<code>'division' or</code>
<code>floating point.h</code>	<code>'underflow' or</code>
	<code>'overflow' or</code>
	<code>'invalid' or</code>
	<code>'all' or</code>
	<code>'common'</code>

Example 1: Set rounding direction to round toward zero, unless the hardware does not support directed rounding modes:

```
INTEGER*4 ieeeer
character*1 mode, out, in
ieeeer = ieee_flags( 'set', 'direction', 'tozero', out )
```

Example 2: Clear rounding direction to default (round toward nearest):

```
character*1 out, in
ieeeer = ieee_flags('clear','direction', in, out )
```

Example 3: Clear all accrued exception-occurred bits:

```
character*18 out
ieeeer = ieee_flags( 'clear', 'exception', 'all', out )
```

Example 4: Detect overflow exception as follows:

```
character*18 out
ieeeer = ieee_flags( 'get', 'exception', 'overflow', out )
if (out .eq. 'overflow' ) stop 'overflow'
```

The above code sets `out` to **overflow** and `ieeeer` to 25 (this value is platform dependent). Similar coding detects exceptions, such as **invalid** or **inexact**.

Example 5: **hand1.f**, write and use a signal handler:

```
external hand
real r / 14.2 /, s / 0.0 /
i = ieee_handler( 'set', 'division', hand )
t = r/s
end

INTEGER*4 function hand ( sig, sip, uap )
```

```

        INTEGER*4 sig, address
        structure /fault/
            INTEGER*4 address
        end structure
        structure /siginfo/
            INTEGER*4 si_signo
            INTEGER*4 si_code
            INTEGER*4 si_errno
            record /fault/ fault
        end structure
        record /siginfo/ sip
        address = sip.fault.address
        write (*,10) address
10    format('Exception at hex address ', z8 )
        end

```

Change the declarations for **address** and **function hand** to **INTEGER\*8** to enable Example 5 in a 64-bit, SPARC environment (**-m64**)

See the *Numerical Computation Guide*. See also: **floatingpoint(3)**, **signal(3)**, **sigfpe(3)**, **floatingpoint(3F)**, **ieee\_flags(3M)**, and **ieee\_handler(3M)**.

### 1.4.26.1 **floatingpoint.h: Fortran IEEE Definitions**

The header file **floatingpoint.h** defines constants and types used to implement standard floating-point according to ANSI/IEEE Std 754-1985.

Include the file in a Fortran 95 source program as follows:

```
#include "floatingpoint.h"
```

Use of this include file requires preprocessing prior to Fortran compilation. The source file referencing this include file will automatically be preprocessed if the name has a **.F**, **.F90** or **.F95** extension.

IEEE Rounding Mode:

<b>fp_direction_type</b>	The type of the IEEE rounding direction mode. The order of enumeration varies according to hardware.
--------------------------	--

SIGFPE Handling:

<b>sigfpe_code_type</b>	The type of a <b>SIGFPE</b> code.
-------------------------	-----------------------------------

<b>sigfpe_handler_type</b>	The type of a user-definable <b>SIGFPE</b> exception handler called to handle a particular <b>SIGFPE</b> code.
<b>SIGFPE_DEFAULT</b>	A macro indicating default <b>SIGFPE</b> exception handling: IEEE exceptions to continue with a default result and to abort for other <b>SIGFPE</b> codes.
<b>SIGFPE_IGNORE</b>	A macro indicating an alternate <b>SIGFPE</b> exception handling, namely to ignore and continue execution.
<b>SIGFPE_ABORT</b>	A macro indicating an alternate <b>SIGFPE</b> exception handling, namely to abort with a core dump.

IEEE Exception Handling:

<b>N_IEEE_EXCEPTION</b>	The number of distinct IEEE floating-point exceptions.
<b>fp_exception_type</b>	The type of the <b>N_IEEE_EXCEPTION</b> exceptions. Each exception is given a bit number.
<b>fp_exception_field_type</b>	The type intended to hold at least <b>N_IEEE_EXCEPTION</b> bits corresponding to the IEEE exceptions numbered by <b>fp_exception_type</b> . Thus, <b>fp_inexact</b> corresponds to the least significant bit and <b>fp_invalid</b> to the fifth least significant bit. Some operations can set more than one exception.

IEEE Classification:

<b>fp_class_type</b>	A list of the classes of IEEE floating-point values and symbols.
----------------------	--

Refer to the *Numerical Computation Guide*. See also **ieee\_environment**(3F).

## 1.4.27 **index, rindex, lnblnk: Index or Length of Substring**

These functions search through a character string:

<b>index(a1, a2)</b>	Index of first occurrence of string <i>a2</i> in string <i>a1</i>
<b>rindex(a1, a2)</b>	Index of last occurrence of string <i>a2</i> in string <i>a1</i>
<b>lnblnk(a1)</b>	Index of last nonblank in string <i>a1</i>

**index** has the following forms:

### 1.4.27.1 **index: First Occurrence of a Substring in a String**

The index is an intrinsic function called by:

---

$n = \text{index}(a1, a2)$

<i>a1</i>	<b>character</b>	Input	Main string
<i>a2</i>	<b>character</b>	Input	Substring
Return value	<b>INTEGER</b>	Output	$n > 0$ : Index of first occurrence of <i>a2</i> in <i>a1</i> $n = 0$ : <i>a2</i> does not occur in <i>a1</i> .

---

If declared **INTEGER\*8**, **index()** will return an **INTEGER\*8** value when compiled for a 64-bit environment and character variable *a1* is a very large character string (greater than 2 Gigabytes).

### 1.4.27.2 **rindex: Last Occurrence of a Substring in a String**

The function is called by:

---

**INTEGER\*4 rindex**

$n = \text{rindex}(a1, a2)$

<i>a1</i>	<b>character</b>	Input	Main string
<i>a2</i>	<b>character</b>	Input	Substring
Return value	<b>INTEGER*4</b> <i>or</i> <b>INTEGER*8</b>	Output	$n > 0$ : Index of last occurrence of <i>a2</i> in <i>a1</i> $n = 0$ : <i>a2</i> does not occur in <i>a1</i> <b>INTEGER*8</b> returned in 64-bit environments

---

### 1.4.27.3 **lbnlkn: Last Nonblank in a String**

The function is called by:

---

$n = \text{lbnlkn}(a1)$

<i>a1</i>	<b>character</b>	Input	String
Return value	<b>INTEGER*4</b> <i>or</i> <b>INTEGER*8</b>	Output	$n > 0$ : Index of last nonblank in <i>a1</i> $n = 0$ : <i>a1</i> is all nonblank <b>INTEGER*8</b> returned in 64-bit environments

---

Example: **index()**, **rindex()**, **lbnlkn()**:



```

demo% cat tindex.f
*
123456789012345678901
character s*24 / 'abcPDQxyz...abcPDQxyz' /
INTEGER*4 declen, index, first, last, len, lnblnk, rindex
declen = len( s )
first = index( s, 'abc' )
last = rindex( s, 'abc' )
lastnb = lnblnk( s )
write(*,*) declen, lastnb
write(*,*) first, last
end
demo% f95 tindex.f
demo% a.out
24 21 <- declen is 24 because intrinsic len() returns the declared length of s
1 13

```

---

**Note** – Programs compiled to run in a 64-bit environment must declare **index**, **rindex** and **lnblnk** (and their receiving variables) **INTEGER\*8** to handle very large character strings.

---

## 1.4.28 inmax: Return Maximum Positive Integer

The function is called by:

---

$m = \text{inmax}()$

Return value	INTEGER*4	Output	The maximum positive integer
--------------	-----------	--------	------------------------------

Example: **inmax**:

```

demo% cat tinmax.f
INTEGER*4 inmax, m
m = inmax()
write(*,*) m
end
demo% f95 tinmax.f
demo% a.out
2147483647
demo%

```

See also **libm\_single(3F)** and **libm\_double(3F)**. See also the non-standard FORTRAN 77 intrinsic function **ephuge()** described in Chapter 3.

## 1.4.29 **itime: Current Time**

**itime** puts the current system time into an integer array: hour, minute, and second. The subroutine is called by:

---

<b>call itime( iarray )</b>			
<i>iarray</i>	<b>INTEGER*4</b>	Output	3-element array: <i>iarray</i> (1) = hour <i>iarray</i> (2) = minute <i>iarray</i> (3) = second

---

Example: **itime**:

```
demo% cat titime.f
      INTEGER*4 iarray(3)
      call itime( iarray )
      write (*, "( ' The time is: ',3i5)" ) iarray
      end
demo% f95 titime.f
demo% a.out
The time is: 15 42 35
```

See also **time(3F)**, **ctime(3F)**, and **fdate(3F)**.

## 1.4.30 **kill: Send a Signal to a Process**

The function is called by:

---

<b>status = kill( pid, signum )</b>			
<i>pid</i>	<b>INTEGER*4</b>	Input	Process ID of one of the user's processes
<i>signum</i>	<b>INTEGER*4</b>	Input	Valid signal number. See <i>signal(3)</i> .
Return value	<b>INTEGER*4</b>	Output	<i>status</i> =0: OK <i>status</i> >0: Error code

---

Example (fragment): Send a message using **kill()**:

```
      INTEGER*4 kill, pid, signum
*      ...
      status = kill( pid, signum )
```

```

    if ( status .ne. 0 ) stop 'kill: error'
    write(*,*) 'Sent signal ', signum, ' to process ', pid
end

```

The function sends signal *signum*, and integer signal number, to the process *pid*. Valid signal numbers are listed in the C include file `/usr/include/sys/signal.h`

See also: `kill(2)`, `signal(3)`, `signal(3F)`, `fork(3F)`, and `perror(3F)`.

## 1.4.31 `link`, `symlink`: Make a Link to an Existing File

`link` creates a link to an existing file. `symlink` creates a symbolic link to an existing file.

The functions are called by:

---

```
status = link( name1, name2 )
```

---

```
INTEGER*4 symlink
```

```
status = symlink( name1, name2 )
```

<i>name1</i>	character*n	Input	Path name of an existing file
<i>name2</i>	character*n	Input	Path name to be linked to the file, <i>name1</i> . <i>name2</i> must not already exist.
Return value	INTEGER*4	Output	<i>status</i> =0: OK <i>status</i> >0: System error code

---

### 1.4.31.1 `link`: Create a Link to an Existing File

Example 1: `link`: Create a link named `data1` to the file, `mlink.db.data.1`:

```

demo% catmlink.f
      character*34 name1/'mlink.db.data.1', name2/'data1'/
      integer*4 link, status
      status = link( name1, name2 )
      if ( status .ne. 0 ) stop 'link: error'
end

demo% f95mlink.f
demo% ls -l data1
data1 not found
demo% a.out
demo% ls -l data1
-rw-rw-r-- 2 generic 2 Aug 11 08:50 data1
demo%

```

## 1.4.31.2 `symlnk`: Create a Symbolic Link to an Existing File

Example 2: `symlnk`: Create a symbolic link named `data1` to the file, `tlink.db.data.1`:

```
demo% cat tsymlnk.f
      character*34 name1/'tlink.db.data.1'/, name2/'data1'/
      INTEGER*4 status, symlnk
      status = symlnk( name1, name2 )
      if ( status .ne. 0 ) stop 'symlnk: error'
      end
demo% f95 tsymlnk.f
demo% ls -l data1
data1 not found
demo% a.out
demo% ls -l data1
lrwxrwxrwx 1 generic 15 Aug 11 11:09 data1 -> tlink.db.data.1
demo%
```

See also: `link(2)`, `symlink(2)`, `perror(3F)`, and `unlink(3F)`.

Note: the path names cannot be longer than `MAXPATHLEN` as defined in `<sys/param.h>`.

## 1.4.32 `loc`: Return the Address of an Object

This intrinsic function is called by:

$k = \text{loc}(arg)$			
<i>arg</i>	Any type	Input	Variable or array
Return value	<b>INTEGER*4</b> -or- <b>INTEGER*8</b>	Output	Address of <i>arg</i>
Returns an <b>INTEGER*8</b> pointer when compiled to run in a 64-bit environment with <b>-m64</b> . See Note below.			

Example: `loc`:

```
INTEGER*4 k, loc
real arg / 9.0 /
k = loc( arg )
write(*,*) k
end
```

---

**Note** – Programs compiled to run in a 64-bit environment should declare **INTEGER\*8** the variable receiving output from the **loc()** function.

---

## 1.4.33 Long, short: Integer Object Conversion

**long** and **short** handle integer object conversions between **INTEGER\*4** and **INTEGER\*2**, and is especially useful in subprogram call lists.

### 1.4.33.1 Long: Convert a Short Integer to a Long Integer

The function is called by:

---

```
call ExpecLong( long(int2) )
```

<i>int2</i>	<b>INTEGER*2</b>	Input
Return value	<b>INTEGER*4</b>	Output

### 1.4.33.2 short: Convert a Long Integer to a Short Integer

The function is:

---

```
INTEGER*2 short
```

```
call ExpecShort( short(int4) )
```

<i>int4</i>	<b>INTEGER*4</b>	Input
Return value	<b>INTEGER*2</b>	Output

Example (fragment): **long()** and **short()**:

```
integer*4 int4/8/, long
integer*2 int2/8/, short
call ExpecLong( long(int2) )
call ExpecShort( short(int4) )
...
end
```

*ExpecLong* is some subroutine called by the user program that expects a *long* (**INTEGER\*4**) integer argument. Similarly, *ExpecShort* expects a *short* (**INTEGER\*2**) integer argument.

**long** is useful if constants are used in calls to library routines and the code is compiled with the **-i2** option.

**short** is useful in similar context when an otherwise long object must be passed as a short integer. Passing an integer to **short** that is too large in magnitude does not cause an error, but will result in unexpected behavior.

## 1.4.34 Longjmp, **isetjmp**: Return to Location Set by **isetjmp**

**isetjmp** sets a location for **longjmp**; **longjmp** returns to that location.

### 1.4.34.1 **isetjmp**: Set the Location for **longjmp**

This intrinsic function is called by:

<i>ival</i> = <b>isetjmp</b> ( <i>env</i> )			
<i>env</i>	<b>INTEGER*4</b>	Output	<i>env</i> is a 12-element integer array. In 64-bit environments it must be declared <b>INTEGER*8</b>
Return value	<b>INTEGER*4</b>	Output	<i>ival</i> = 0 if <b>isetjmp</b> is called explicitly <i>ival</i> ≠ 0 if <b>isetjmp</b> is called through <b>longjmp</b>

### 1.4.34.2 **longjmp**: Return to the Location Set by **isetjmp**

The subroutine is called by:

call <b>longjmp</b> ( <i>env</i> , <i>ival</i> )			
<i>env</i>	<b>INTEGER*4</b>	Input	<i>env</i> is the 12-word integer array initialized by <b>isetjmp</b> . In 64-bit environments it must be declared <b>INTEGER*8</b>
<i>ival</i>	<b>INTEGER*4</b>	Output	<i>ival</i> = 0 if <b>isetjmp</b> is called explicitly <i>ival</i> ≠ 0 if <b>isetjmp</b> is called through <b>longjmp</b>

## Description

The **isetjmp** and **longjmp** routines are used to deal with errors and interrupts encountered in a low-level routine of a program. They are **f95** intrinsics.

These routines should be used only as a last resort. They require discipline, and are not portable. Read the man page, **setjmp**(3V), for bugs and other details.

**isetjmp** saves the stack environment in *env*. It also saves the register environment.

**longjmp** restores the environment saved by the last call to **isetjmp**, and returns in such a way that execution continues as if the call to **isetjmp** had just returned the value *ival*.

The integer expression *ival* returned from **isetjmp** is zero if **longjmp** is not called, and nonzero if **longjmp** is called.

Example: Code fragment using **isetjmp** and **longjmp**:

```

INTEGER*4  env(12)
common /jmpblk/ env
j = isetjmp( env )
if ( j .eq. 0 ) then
    call sbrtnA
else
    call error_processor
end if
end
subroutine sbrtnA
INTEGER*4  env(12)
common /jmpblk/ env
call longjmp( env, ival )
return
end

```

## Restrictions

- You must invoke **isetjmp** before calling **longjmp**.
- The *env* integer array argument to **isetjmp** and **longjmp** must be at least 12 elements long.
- You must pass the *env* variable from the routine that calls **isetjmp** to the routine that calls **longjmp**, either by common or as an argument.
- **longjmp** attempts to clean up the stack. **longjmp** must be called from a lower call-level than **isetjmp**.
- Passing **isetjmp** as an argument that is a procedure name does not work.

See **setjmp**(3V).

## 1.4.35 malloc, malloc64, realloc, free: Allocate/Reallocate/Deallocate Memory

The functions **malloc()**, **malloc64()**, and **realloc()** allocate blocks of memory and return the starting address of the block. The return value can be used to set an **INTEGER** or Cray-style **POINTER** variable. **realloc()** reallocates an existing memory block with a new size. **free()** deallocates memory blocks allocated by **malloc()**, **malloc64()**, or **realloc()**.

---

**Note** – These routines are implemented as intrinsic functions in **f95**, but were external functions in **f77**. They should not appear on type declarations in Fortran 95 programs, or on **EXTERNAL** statements unless you wish to use your own versions. The **realloc()** routine is only implemented for **f95**.

---

Standard-conforming Fortran 95 programs should use **ALLOCATE** and **DEALLOCATE** statements on **ALLOCATABLE** arrays to perform dynamic memory management, and not make direct calls to **malloc/realloc/free**.

Legacy Fortran 77 programs could use **malloc()/malloc64()** to assign values to Cray-style **POINTER** variables, which have the same data representation as **INTEGER** variables. Cray-style **POINTER** variables are implemented in **f95** to support portability from Fortran 77.

### 1.4.35.1 Allocate Memory: **malloc**, **malloc64**

The **malloc()** function is called by:

$k = \text{malloc}(n)$			
$n$	<b>INTEGER</b>	Input	Number of bytes of memory
Return value	<b>INTEGER(Cray POINTER)</b>	Output	$k > 0$ : $k$ = address of the start of the block of memory allocated $k = 0$ : Error
An <b>INTEGER*8</b> pointer value is returned when compiled for a 64-bit environment with <b>-m64</b> . See Note below.			

---

**Note** – This function is intrinsic in Fortran 95 and was external in Fortran 77. Fortran 77 programs compiled to run in 64-bit environments would declare the **malloc()** function and the variables receiving its output as **INTEGER\*8**. The function **malloc64(3F)** was provided to make programs portable between 32-bit and 64-bit environments.

$k = \text{malloc64}(n)$			
$n$	<b>INTEGER*8</b>	Input	Number of bytes of memory
Return value	<b>INTEGER*8</b> (Cray <b>POINTER</b> )	Output	$k > 0$ : $k$ = address of the start of the block of memory allocated $k = 0$ : Error



These functions allocate an area of memory and return the address of the start of that area. (In a 64-bit environment, this returned byte address may be outside the **INTEGER\*4** numerical range—the receiving variables must be declared **INTEGER\*8** to avoid truncation of the memory address.) The region of memory is not initialized in any way, and it should not be assumed to be preset to anything, especially zero!

Example: Code fragment using **malloc()**:

```

parameter (NX=1000)
integer ( p2X, X )
real*4 X(1)
...
p2X = malloc( NX*4 )
if ( p2X .eq. 0 ) stop 'malloc: cannot allocate'
do 11 i=1,NX
11      X(i) = 0.
...
end

```

In the above example, we acquire 4,000 bytes of memory, pointed to by **p2X**, and initialize it to zero.

## 1.4.35.2 Reallocate Memory: **realloc**

The **realloc()** **f95** intrinsic function is called by:

```
k = realloc(ptr, n)
```

<i>ptr</i>	<b>INTEGER</b>	Input	Pointer to existing memory block. (Value returned from a previous <b>malloc()</b> or <b>realloc()</b> call).
<i>n</i>	<b>INTEGER</b>	Input	Requested new size of block, in bytes.
Return value	<b>INTEGER</b> (Cray <b>POINTER</b> )	Output	<i>k</i> >0: <i>k</i> =address of the start of the new block of memory allocated <i>k</i> =0: Error
An <b>INTEGER*8</b> pointer value is returned when compiled for a 64-bit environment with <b>-m64</b> . See Note below.			

The **realloc()** function changes the size of the memory block pointed to by **ptr** to **n** bytes and returns a pointer to the (possibly moved) new block. The contents of the memory block will be unchanged up to the lesser of the new and old sizes.

If **ptr** is zero, **realloc()** behaves the same as **malloc()** and allocates a new memory block of size **n** bytes.

If **n** is zero and **ptr** is not zero, the memory block pointed to is made available for further allocation and is returned to the system only upon termination of the application.

Example: Using **malloc()** and **realloc()** and Cray-style **POINTER** variables:

```

PARAMETER (nsize=100001)
POINTER (p2space,space)
REAL*4 space(1)

p2space = malloc(4*nsize)
if(p2space .eq. 0) STOP 'malloc: cannot allocate space'
...
p2space = realloc(p2space, 9*4*nsize)
if(p2space .eq. 0) STOP 'realloc: cannot reallocate space'
...
CALL free(p2space)
...

```

Note that **realloc()** is only implemented for **f95**.

### 1.4.35.3 **free: Deallocate Memory Allocated by Malloc**

The subroutine is called by:

<b>call free ( ptr )</b>		
<i>ptr</i>	Cray <b>POINTER</b>	Input

**free** deallocates a region of memory previously allocated by **malloc** and **realloc()**. The region of memory is returned to the memory manager; it is no longer available to the user's program.

Example: **free()**:

```

real x
pointer ( ptr, x )
ptr = malloc ( 10000 )
call free ( ptr )
end

```

### 1.4.36 **mvbits: Move a Bit Field**

The subroutine is called by:

---

**call mvbits** ( *src*, *ini1*, *nbits*, *des*, *ini2* )

<i>src</i>	<b>INTEGER*4</b>	Input	Source
<i>ini1</i>	<b>INTEGER*4</b>	Input	Initial bit position in the source
<i>nbits</i>	<b>INTEGER*4</b>	Input	Number of bits to move
<i>des</i>	<b>INTEGER*4</b>	Output	Destination
<i>ini2</i>	<b>INTEGER*4</b>	Input	Initial bit position in the destination

Example: **mvbits**:

```
demo% cat mvb1.f
* mvb1.f -- From src, initial bit 0, move 3 bits to des, initial *      bit 3.
*   src   des
* 543210 543210      Bit numbers
* 000111 000001      Values before move
* 000111 111001      Values after move
      INTEGER*4 src, ini1, nbits, des, ini2
      data src, ini1, nbits, des, ini2
      1 / 7, 0, 3, 1, 3 /
      call mvbits ( src, ini1, nbits, des, ini2 )
      write (*,"(5o3)") src, ini1, nbits, des, ini2
      end
demo% f95 mvb1.f
demo% a.out
 7 0 3 71 3
demo%
```

Note the following:

- Bits are numbered 0 to 31, from least significant to most significant.
- **mvbits** changes only bits *ini2* through *ini2+nbits-1* of the *des* location, and no bits of the *src* location.
- The restrictions are:
  - $ini1 + nbits \geq 32$
  - $ini2 + nbits \leq 32$

## 1.4.37 **perror, gerror, ierrno: Get System Error Messages**

These routines perform the following functions:

<b>perror</b>	Print a message to Fortran logical unit 0, <b>stderr</b> .
<b>gerror</b>	Get a system error message (of the last detected system error)
<b>ierarno</b>	Get the error number of the last detected system error.

### 1.4.37.1 **perror: Print Message to Logical Unit 0, stderr**

The subroutine is called by:

---

**call perror( *string* )**

<i>string</i>	<b>character*n</b>	Input	The message. It is written preceding the standard error message for the last detected system error.
---------------	--------------------	-------	---

Example 1:

```
call perror( "file is for formatted I/O" )
```

### 1.4.37.2 **gerror: Get Message for Last Detected System Error**

The subroutine or function is called by:

---

**call gerror( *string* )**

<i>string</i>	<b>character*n</b>	Output	Message for the last detected system error
---------------	--------------------	--------	--

Example 2: **gerror()** as a subroutine:

```
character string*30
...
call gerror ( string )
write(*,*) string
```

Example 3: **gerror()** as a function; *string* not used:

```
character gerror*30, z*30
...
z = gerror ( )
write(*,*) z
```

### 1.4.37.3 **ierarno: Get Number for Last Detected System Error**

The function is called by:

---

`n = ierrno()`

Return value	INTEGER*4	Output	Number of last detected system error
--------------	-----------	--------	--------------------------------------

This number is updated only when an error actually occurs. Most routines and I/O statements that might generate such errors return an error code after the call; that value is a more reliable indicator of what caused the error condition.

Example 4: `ierrno()`:

```

      INTEGER*4 ierrno, n
      ...
      n = ierrno()
      write(*,*) n

```

See also `intro(2)` and `perorr(3)`.

Note:

- `string` in the call to `perorr` cannot be longer than 127 characters.
- The length of the string returned by `perorr` is determined by the calling program.
- Runtime I/O error codes for `f95` are listed in the *Fortran User's Guide*.

## 1.4.38 `putc, fputc`: Write a Character to a Logical Unit

`putc` writes to logical unit 6, normally the control terminal output.

`fputc` writes to a logical unit.

These functions write a character to the file associated with a Fortran logical unit bypassing normal Fortran I/O.

Do not mix normal Fortran output with output by these functions on the same unit.

Note that to write any of the special `\` escape characters, such as `'\n'` newline, requires compiling with `-f77=backslash` FORTRAN 77 compatibility option.

### 1.4.38.1 `putc`: Write to Logical Unit 6

The function is called by:

---

```

      INTEGER*4 putc
      status = putc( char )

```

---

<i>char</i>	<b>character</b>	Input	The character to write to the unit
Return value	<b>INTEGER*4</b>	Output	<i>status</i> =0: OK <i>status</i> >0: System error code

Example: **putc()**:

```
demo% cat tputc.f
      character char, s*10 / 'OK by putc' /
      INTEGER*4 putc, status
      do i = 1, 10
        char = s(i:i)
        status = putc( char )
      end do
      status = putc( '\n' )
      end
demo% f95 -f77=backslash tputc.f
demo% a.out
OK by putc
demo%
```

## 1.4.38.2 fputc: Write to Specified Logical Unit

The function is called by:

<b>INTEGER*4 fputc</b>			
<i>status</i> = <b>fputc</b> ( <i>lunit</i> , <i>char</i> )			
<i>lunit</i>	<b>INTEGER*4</b>	Input	The unit to write to
<i>char</i>	<b>character</b>	Input	The character to write to the unit
Return value	<b>INTEGER*4</b>	Output	<i>status</i> =0: OK <i>status</i> >0: System error code

Example: **fputc()**:

```
demo% cat tfputc.f
      character char, s*11 / 'OK by fputc' /
      INTEGER*4 fputc, status
      open( 1, file='tfputc.data' )
      do i = 1, 11
        char = s(i:i)
        status = fputc( 1, char )
      end do
```

```

        status = fputc( 1, '\n' )
    end
demo% f95 -f77=backslash tfputc.f
demo% a.out
demo% cat tfputc.data
OK by fputc
demo%

```

See also `putc(3S)`, `intro(2)`, and `perror(3F)`.

## 1.4.39 `qsort`, `qsort64`: Sort the Elements of a One-Dimensional Array

The subroutine is called by:

---

```
call qsort( array, len, isize, compar )
```

```
call qsort64( array, len8, isize8, compar )
```

---

<i>array</i>	<b>array</b>	Input	Contains the elements to be sorted
<i>len</i>	<b>INTEGER*4</b>	Input	Number of elements in the array.
<i>len8</i>	<b>INTEGER*8</b>	Input	Number of elements in the array
<i>isize</i>	<b>INTEGER*4</b>	Input	Size of an element, typically: 4 for integer or real 8 for double precision or complex 16 for double complex Length of character object for character arrays
<i>isize8</i>	<b>INTEGER*8</b>	Input	Size of an element, typically: 4_8 for integer or real 8_8 for double precision or complex 16_8 for double complex Length of character object for character arrays
<i>compar</i>	function name	Input	Name of a user-supplied <b>INTEGER*2</b> function. Determines sorting order: <code>compar(arg1,arg2)</code>

---

Use `qsort64` in 64-bit environments with arrays larger than 2 Gbytes. Be sure to specify the array length, *len8*, and the element size, *isize8*, as **INTEGER\*8** data. Use the Fortran 95 style constants to explicitly specify **INTEGER\*8** constants.

The `compar`(*arg1*, *arg2*) arguments are elements of *array*, returning:

Negative	If <i>arg1</i> is considered to precede <i>arg2</i>
Zero	If <i>arg1</i> is equivalent to <i>arg2</i>
Positive	If <i>arg1</i> is considered to follow <i>arg2</i>

For example:

```
demo% cat tqsort.f
      external compar
      integer*2 compar
      INTEGER*4 array(10)/5,1,9,0,8,7,3,4,6,2/, len/10/,
1      isize/4/
      call qsort( array, len, isize, compar )
      write(*,'(10i3)') array
      end
      integer*2 function compar( a, b )
      INTEGER*4 a, b
      if ( a .lt. b ) compar = -1
      if ( a .eq. b ) compar = 0
      if ( a .gt. b ) compar = 1
      return
      end
demo% f95 tqsort.f
demo% a.out
  0 1 2 3 4 5 6 7 8 9
```

## 1.4.40 ran: Generate a Random Number Between 0 and 1

Repeated calls to `ran` generate a sequence of random numbers with a uniform distribution. See `lcrans`(3m).

$r = \text{ran}(i)$			
<i>i</i>	INTEGER*4	Input	Variable or array element
<i>r</i>	REAL	Output	Variable or array element

Example: `ran`:

```
demo% cat ran1.f
* ran1.f -- Generate random numbers.
      INTEGER*4 i, n
```



```

      real r(10)
      i = 760013
      do n = 1, 10
         r(n) = ran ( i )
      end do
      write ( *, "( 5 f11.6 )" ) r
      end
demo% f95 ran1.f
demo% a.out
      0.222058 0.299851 0.390777 0.607055 0.653188
      0.060174 0.149466 0.444353 0.002982 0.976519
demo%
```

Note the following:

- The range includes 0.0 and excludes 1.0.
- The algorithm is a multiplicative, congruential type, general random number generator.
- In general, the value of *i* is set *once* during execution of the calling program.
- The initial value of *i* should be a large odd integer.
- Each call to **RAN** gets the next random number in the sequence.
- To get a different sequence of random numbers each time you run the program, you must set the argument to a different initial value for each run.
- The argument is used by **RAN** to store a value for the calculation of the next random number according to the following algorithm:

$$\text{SEED} = 6909 * \text{SEED} + 1 \pmod{2^{**}32}$$

- **SEED** contains a 32-bit number, and the high-order 24 bits are converted to floating point, and that value is returned.

## 1.4.41 rand, drand, irand: Return Random Values

**rand** returns real values in the range 0.0 through 1.0.

**drand** returns double precision values in the range 0.0 through 1.0.

**irand** returns positive integers in the range 0 through 2147483647.

These functions use **random(3)** to generate sequences of random numbers. The three functions share the same 256 byte state array. The only advantage of these functions is that they are widely available on UNIX systems. For better random number generators, compare **lcrans**, **addrans**, and **shufrans**. See also **random(3)**, and the *Numerical Computation Guide*

---

$i = \mathbf{irand}(k)$

$r = \mathbf{rand}(k)$

$d = \mathbf{drand}(k)$

$k$	<b>INTEGER*4</b>	Input	$k=0$ : Get next random number in the sequence $k=1$ : Restart sequence, return first number $k>0$ : Use as a seed for new sequence, return first number
<b>rand</b>	<b>REAL*4</b>	Output	
<b>drand</b>	<b>REAL*8</b>	Output	
<b>irand</b>	<b>INTEGER*4</b>	Output	

Example: **irand()**:

demo% **cat trand.f**

```
integer*4 v(5), iflag/0/
do i = 1, 5
  v(i) = irand( iflag )
end do
write(*,*) v
end
```

demo% **f95 trand.f**

demo% **a.out**

```
2078917053 143302914 1027100827 1953210302 755253631
```

demo%

## 1.4.42 rename: Rename a File

The function is called by:

---

**INTEGER\*4 rename**

$status = \mathbf{rename}(from, to)$

$from$	<b>character*n</b>	Input	Path name of an existing file
$to$	<b>character*n</b>	Input	New path name for the file

Return value	<b>INTEGER*4</b>	Output	<i>status</i> =0: OK <i>status</i> >0: System error code
--------------	------------------	--------	---

If the file specified by *to* exists, then both *from* and *to* must be the same type of file, and must reside on the same file system. If *to* exists, it is removed first.

Example: **rename()**—Rename file **trename.old** to **trename.new**

```
demo% cat trename.f
      INTEGER*4 rename, status
      character*18 from/'trename.old'/, to/'trename.new'/
      status = rename( from, to )
      if ( status .ne. 0 ) stop 'rename: error'
      end
demo% f95 trename.f
demo% ls trename*
trename.f trename.old
demo% a.out
demo% ls trename*
trename.f trename.new
demo%
```

See also **rename(2)** and **perror(3F)**.

Note: the path names cannot be longer than **MAXPATHLEN** as defined in **<sys/param.h>**.

## 1.4.43 secnds: Get System Time in Seconds, Minus Argument

```
t = secnds( t0 )
```

<i>t0</i>	<b>REAL</b>	Input	Constant, variable, or array element
Return Value	<b>REAL</b>	Output	Number of seconds since midnight, minus <i>t0</i>

Example: **secnds**:

```
demo% cat sec1.f
      real elapsed, t0, t1, x, y
      t0 = 0.0
      t1 = secnds( t0 )
      y = 0.1
      do i = 1, 10000
         x = asin( y )
      end do
```

```

        elapsed = secnds( t1 )
        write ( *, 1 ) elapsed
1     format ( ' 10000 arcsines: ', f12.6, ' sec' )
        end
demo% f95 sec1.f
demo% a.out
10000 arcsines:      0.009064 sec
demo%
```

Note that:

- The returned value from **SECNDS** is accurate to 0.01 second.
- The value is the system time, as the number of seconds from midnight, and it correctly spans midnight.
- Some precision may be lost for small time intervals near the end of the day.

## 1.4.44 set\_io\_err\_handler, get\_io\_err\_handler: Set and Get I/O Error Handler

**set\_io\_err\_handler()** declares a user-defined routine to be called whenever errors are detected on a specified input logical unit.

**get\_io\_err\_handler()** returns the address of the currently declared error handling routine.

These routines are module subroutines and can only be accessed when **USE SUN\_IO\_HANDLERS** appears in the calling routine.

---

**USE SUN\_IO\_HANDLERS**

**call set\_io\_err\_handler**(*iu*, *subr\_name*, *istat*)

<i>iu</i>	<b>INTEGER*8</b>	Input	Logical unit number
<i>subr_name</i>	<b>EXTERNAL</b>	Input	Name of user-supplied error handler subroutine.
<i>istat</i>	<b>INTEGER*4</b>	Output	Return status.

---

**USE SUN\_IO\_HANDLERS**

**call get\_io\_err\_handler**(*iu*, *subr\_pointer*, *istat*)

<i>iu</i>	<b>INTEGER*8</b>	Input	Logical unit number
-----------	------------------	-------	---------------------

<i>subr_pointer</i>	<b>POINTER</b>	Output	Address of currently declared handler routine.
<i>istat</i>	<b>INTEGER*4</b>	Output	Return status.

**SET\_IO\_ERR\_HANDLER** sets the user-supplied subroutine *subr\_name* to be used as the I/O error handler for the logical unit *iu* when an input error occurs. *iu* has to be a connected Fortran logical unit for a formatted file. *istat* will be set to a non-zero value if there is an error, otherwise it is set to zero.

For example, if **SET\_IO\_ERR\_HANDLER** is called before the logical unit *iu* has been opened, *istat* will be set to 1001 ("Illegal Unit"). If *subr\_name* is NULL user error-handling is turned off and the program reverts to default Fortran error handling.

Use **GET\_IO\_ERR\_HANDLER** to get the address of the function currently being used as the error handler for this logical unit. For example, call **GET\_IO\_ERR\_HANDLER** to save the current I/O before switching to another handler routine. The error handler can be restored with the saved value later.

*subr\_name* is the name of the user-supplied routine to handle the I/O error on logical unit *iu*. The runtime I/O library passes all relevant information to *subr\_name*, enabling this routine to diagnose the problem and possibly fix the error before continuing.

The interface to the user-supplied error handler routine is as follows:

```

SUBROUTINE SUB_NAME(UNIT, SRC_FILE, SRC_LINE, DATA_FILE, FILE_POS,
CURR_BUFF, CURR_ITEM, CORR_CHAR, CORR_ACTION )
  INTENT (IN) UNIT, SRC_FILE, SRC_LINE, DATA_FILE
  INTENT (IN) FILE_POS, CURR_BUFF, CURR_ITEM
  INTENT (OUT) CORR_CHAR, CORR_ACTION

```

<b>UNIT</b>	<b>INTEGER*8</b>	Input	Logical unit number of the input file reporting an error.
<b>SRC_FILE</b>	<b>CHARACTER*(*)</b>	Input	Name of the Fortran source file originating the input operation.
<b>SRC_LINE</b>	<b>INTEGER*8</b>	Input	Line number in <b>SRC_FILE</b> of the input operation with an error.

<b>DATA_FILE</b>	<b>CHARACTER*(*)</b>	Input	Name of the data file being read. Available only if the file is an opened external file. If the name is not available, (logical unit 5 for example), <b>DATA_FILE</b> is set to a zero-length character data item.
<b>FILE_POS</b>	<b>INTEGER*8</b>	Input	Current position in the input file, in bytes. Defined only if the name of the <b>DATA_FILE</b> is known.
<b>CURR_BUFF</b>	<b>CHARACTER*(*)</b>	Input	Character string containing the remaining data from the input record. The bad input character is the first character in the string.
<b>CURR_ITEM</b>	<b>INTEGER*8</b>	Input	The number of input items in the record that have been read, including the current one, when the error is detected. For example: <b>READ(12,10)L,(ARR(I),I=1,L)</b>  IF the value of <b>CURR_ITEM</b> is 15 in this case, it means the error happens while reading the 14th element of <b>ARR</b> , with <b>L</b> being the first item and <b>ARR(1)</b> being the second, and so on.
<b>CORR_CHAR</b>	<b>CHARACTER</b>	Output	The user-supplied corrected character to be returned by the handler. This value is used only when <b>CORR_ACTION</b> is non-zero. If <b>CORR_CHAR</b> is an invalid character, the handler will be called again until a valid character is returned. This could cause an infinite loop, a situation the user is required to protect against.
<b>CORR_ACTION</b>	<b>INTEGER</b>	Output	Specifies the corrective action to be taken by the I/O library. With a zero value no special action is taken and the library reverts to its default error processing. A value of <b>1</b> returns <b>CORR_CHAR</b> to the I/O error processing routine.

### 1.4.44.1 Limitations

The I/O handler can only replace once character with another character. It cannot replace one character with more than one character.

The error recovery algorithm can only fix a bad character it currently reads, not a bad character which has already been interpreted as a valid character in a different context. For example, in a list-directed read, if the input is "1.234509.8765" when the correct input should be "1.2345 9.8765" then the I/O library will run into an error at the second period because it is not a valid number. But, by that time, it is not possible to go back and change the '0' into a blank.

Currently, this error-handling capability does not work for namelist-directed input. When doing namelist-directed input, the specified I/O error handler will not be invoked when an error occurs.

I/O error handlers can only be set for external files, not internal files, because there are no logical units associated with internal files.

The I/O error handler is called only for syntactic errors, not system errors or semantic errors (such as an overflowed input value).

It is possible to have an infinite loop if the user-supplied I/O error handler keeps providing a bad character to the I/O library, causing it to call the user-supplied I/O error handler over and over again. If an error keeps occurring at the same file position then the error handler should terminate itself. One way to do this is to take the default error path by setting **CORR\_ACTION** to 0. Then the I/O library will continue with the normal error handling.

## 1.4.45 **sh: Fast Execution of an sh Command**

The function is called by:

---

**INTEGER\*4 sh**

*status* = **sh**( *string* )

<i>string</i>	<b>character*n</b>	Input	String containing command to do
Return value	<b>INTEGER*4</b>	Output	Exit status of the shell executed. See <i>wait(2)</i> for an explanation of this value.

---

Example: **sh()**:

```

character*18 string / 'ls > MyOwnFile.names' /
INTEGER*4 status, sh
status = sh( string )
if ( status .ne. 0 ) stop 'sh: error'
...
end

```

The function **sh** passes *string* to the **sh** shell as input, as if the string had been typed as a command.

The current process waits until the command terminates.

The functions **sh(3f)** and **system(3f)** pass the argument string to a shell for execution. They convert the argument string from a Fortran character value to a C string value and pass it to the C routine **system(3c)**. The routines **sh(3f)** and **system(3f)** differ in that **system** flushes the

Fortran I/O buffers before calling the C routine system, while **sh** does not. Flushing the buffers can take significant time, and so, if any Fortran output is irrelevant to the result of the call, the routine **sh** is preferred over the routine **system**.

The **sh()** function is not MT-safe. Do not call it from multithreaded or parallelized programs.

See also: **execve(2)**, **wait(2)**, and **system(3c)**.

Note: *string* cannot be longer than 1,024 characters.

## 1.4.46 **signal: Change the Action for a Signal**

The function is called by:

<b>INTEGER*4 signal or INTEGER*8 signal</b>			
<i>n</i> = <b>signal</b> ( <i>signum</i> , <i>proc</i> , <i>flag</i> )			
<i>signum</i>	<b>INTEGER*4</b>	Input	Signal number; see <i>signal(3)</i>
<i>proc</i>	Routine name	Input	Name of user signal handling routine; must be in an external statement
<i>flag</i>	<b>INTEGER*4</b>	Input	<i>flag</i> < 0: Use <i>proc</i> as the signal handler <i>flag</i> ≥ 0: Ignore <i>proc</i> ; pass <i>flag</i> as the action: <i>flag</i> = 0: Use the default action <i>flag</i> = 1: Ignore this signal
Return value	<b>INTEGER*4</b>	Output	<i>n</i> = -1: System error <i>n</i> > 0: Definition of previous action <i>n</i> > 1: <i>n</i> = Address of routine that would have been called <i>n</i> < -1: If <i>signum</i> is a valid signal number, then: <i>n</i> = address of routine that would have been called. If <i>signum</i> is a <i>not</i> a valid signal number, then: <i>n</i> is an error number.
	<b>INTEGER*8</b>		On 64-bit environments, <b>signal</b> and the variables receiving its output must be declared <b>INTEGER*8</b>

If *proc* is called, it is passed the signal number as an integer argument.

If a process incurs a signal, the default action is usually to clean up and abort. A signal handling routine provides the capability of catching specific exceptions or interrupts for special processing.

The returned value can be used in subsequent calls to **signal** to restore a previous action definition.



You can get a negative return value even though there is no error. In fact, if you pass a *valid* signal number to `signal()`, and you get a return value less than -1, then it is OK.

`floatingpoint.h` defines *proc* values `SIGFPE_DEFAULT`, `SIGFPE_IGNORE`, and `SIGFPE_ABORT`. See “1.4.26.1 `floatingpoint.h`: Fortran IEEE Definitions” on page 54.

In 64-bit environments, `signal` must be declared `INTEGER*8`, along with the variables receiving its output, to avoid truncation of the address that may be returned.

See also `kill(1)`, `signal(3)`, and `kill(3F)`, and *Numerical Computation Guide*.

## 1.4.47 `sleep`: Suspend Execution for an Interval

The subroutine is called by:

---

call `sleep( itime )`

---

<i>itime</i>	<code>INTEGER*4</code>	Input	Number of seconds to sleep
--------------	------------------------	-------	----------------------------

---

The actual time can be up to 1 second less than *itime* due to granularity in system timekeeping.

Example: `sleep()`:

```

INTEGER*4 time / 5 /
write(*,*) 'Start'
call sleep( time )
write(*,*) 'End'
end

```

See also `sleep(3)`.

## 1.4.48 `stat`, `lstat`, `fstat`: Get File Status

These functions return the following information:

- device,
- inode number,
- protection,
- number of hard links,
- user ID,
- group ID,
- device type,
- size,
- access time,

- modify time,
- status change time,
- optimal blocksize,
- blocks allocated

Both **stat** and **lstat** query by file name. **fstat** queries by logical unit.

### 1.4.48.1 **stat: Get Status for File, by File Name**

The function is called by:

---

**INTEGER\*4 stat**

*ierr = stat ( name, statb )*

<i>name</i>	<b>character*n</b>	Input	Name of the file
<i>statb</i>	<b>INTEGER*4</b>	Output	Status structure for the file, 13-element array
Return value	<b>INTEGER*4</b>	Output	<i>ierr</i> =0: OK <i>ierr</i> >0: Error code

Example 1: **stat()**:

```

character name*18 /'MyFile'/
INTEGER*4 ierr, stat, lunit/1/, statb(13)
open( unit=lunit, file=name )
ierr = stat ( name, statb )
if ( ierr .ne. 0 ) stop 'stat: error'
write(*,*)'UID of owner = ',statb(5),'
1  blocks = ',statb(13)
end

```

### 1.4.48.2 **fstat: Get Status for File, by Logical Unit**

The function

---

**INTEGER\*4 fstat**

*ierr = fstat ( lunit, statb )*

<i>lunit</i>	<b>INTEGER*4</b>	Input	Logical unit number
<i>statb</i>	<b>INTEGER*4</b>	Output	Status for the file: 13-element array

Return value	<b>INTEGER*4</b>	Output	<i>ierr</i> =0: OK <i>ierr</i> >0: Error code
--------------	------------------	--------	--

is called by:

Example 2: **fstat()**:

```

character name*18 /'MyFile'/
INTEGER*4 fstat, lunit/1/, statb(13)
open( unit=lunit, file=name )
ierr = fstat ( lunit, statb )
if ( ierr .ne. 0 ) stop 'fstat: error'
write(*,*)'UID of owner = ',statb(5),',
1   blocks = ',statb(13)
end

```

### 1.4.48.3

## **lstat: Get Status for File, by File Name**

The function is called by:

*ierr* = **lstat** ( *name*, *statb* )

<i>name</i>	<b>character*n</b>	Input	File name
<i>statb</i>	<b>INTEGER*4</b>	Output	Status array of file, 13 elements
Return value	<b>INTEGER*4</b>	Output	<i>ierr</i> =0: OK <i>ierr</i> >0: Error code

Example 3: **lstat()**:

```

character name*18 /'MyFile'/
INTEGER*4 lstat, lunit/1/, statb(13)
open( unit=lunit, file=name )
ierr = lstat ( name, statb )
if ( ierr .ne. 0 ) stop 'lstat: error'
write(*,*)'UID of owner = ',statb(5),',
1   blocks = ',statb(13)
end

```

### 1.4.48.4

## **Detail of Status Array for Files**

The meaning of the information returned in the **INTEGER\*4** array *statb* is as described for the structure *stat* under **stat(2)**.

Spare values are not included. The order is shown in the following table:

<b>statb(1)</b>	Device inode resides on
<b>statb(2)</b>	This inode's number
<b>statb(3)</b>	Protection
<b>statb(4)</b>	Number of hard links to the file
<b>statb(5)</b>	User ID of owner
<b>statb(6)</b>	Group ID of owner
<b>statb(7)</b>	Device type, for inode that is device
<b>statb(8)</b>	Total size of file
<b>statb(9)</b>	File last access time
<b>statb(10)</b>	File last modify time
<b>statb(11)</b>	File last status change time
<b>statb(12)</b>	Optimal blocksize for file system I/O ops
<b>statb(13)</b>	Actual number of blocks allocated

See also **stat(2)**, **access(3F)**, **perorr(3F)**, and **time(3F)**.

Note: the path names can be no longer than **MAXPATHLEN** as defined in `<sys/param.h>`.

## 1.4.49 stat64, lstat64, fstat64: Get File Status

64-bit "long file" versions of **stat**, **lstat**, **fstat**. These routines are identical to the non-64-bit routines, except that the 13-element array *statb* must be declared **INTEGER\*8**.

## 1.4.50 system: Execute a System Command

The function is called by:

<b>INTEGER*4 system</b>			
<i>status</i> = <b>system</b> ( <i>string</i> )			
<i>string</i>	<b>character*n</b>	Input	String containing command to do
Return value	<b>INTEGER*4</b>	Output	Exit status of the shell executed. See <i>wait(2)</i> for an explanation of this value.

Example: **system()**:

```

character*8 string / 'ls s*' /
INTEGER*4 status, system
status = system( string )
if ( status .ne. 0 ) stop 'system: error'
end

```

The function **system** passes *string* to your shell as input, as if the string had been typed as a command. Note: *string* cannot be longer than 1024 characters.

If **system** can find the environment variable **SHELL**, then **system** uses the value of **SHELL** as the command interpreter (shell); otherwise, it uses **sh(1)**.

The current process waits until the command terminates.

Historically, **cc** developed with different assumptions:

- If **cc** calls **system**, the shell is always the Bourne shell.

The **system** function flushes all open files:

- For output files, the buffer is flushed to the actual file.
- For input files, the position of the pointer is unpredictable.

The functions **sh(3f)** and **system(3f)** pass the argument string to a shell for execution. They convert the argument string from a Fortran character value to a C string value and pass it to the C routine **system(3c)**. The routines **sh(3f)** and **system(3f)** differ in that **system** flushes the Fortran I/O buffers before calling the C routine **system**, while **sh** does not. Flushing the buffers can take significant time, and so, if any Fortran output is irrelevant to the result of the call, the routine **sh** is preferred over the routine **system**.

See also: **execve(2)**, **wait(2)**, and **system(3)**.

The **system()** function is not MT-safe. Do not call it from multithreaded or parallelized programs.

## 1.4.51 **time, ctime, ltime, gmtime: Get System Time**

These routines have the following functions:

<b>time</b>	Standard version: Get system time as integer (seconds since 0 GMT 1/1/70) VMS Version: Get the system time as character (hh:mm:ss)
<b>ctime</b>	Convert a system time to an ASCII string.
<b>ltime</b>	Dissect a system time into month, day, and so forth, local time.

<b>gmtime</b>	Dissect a system time into month, day, and so forth, GMT.
---------------	---

### 1.4.51.1 **time: Get System Time**

The `time()` function is called by:

**INTEGER\*4 time** or **INTEGER\*8**

*n = time()* Standard Version

Return value	<b>INTEGER*4</b>	Output	Time, in seconds, since 0:0:0, GMT, 1/1/70
	<b>INTEGER*8</b>	Output	In 64-bit environments, <b>time</b> returns an <b>INTEGER*8</b> value

The function **time()** returns an integer with the time since 00:00:00 GMT, January 1, 1970, measured in seconds. This is the value of the operating system clock.

Example: **time()**, version standard with the operating system:

```
demo% cat ttime.f
      INTEGER*4 n, time
      n = time()
      write(*,*) 'Seconds since 0 1/1/70 GMT = ', n
      end
demo% f95 ttime.f
demo% a.out
Seconds since 0 1/1/70 GMT = 913240205
demo%
```

### 1.4.51.2 **ctime: Convert System Time to Character**

The function **ctime** converts a system time, *stime*, and returns it as a 24-character ASCII string.

The function is called by:

**CHARACTER ctime\*24**

*string = ctime( stime )*

<i>stime</i>	<b>INTEGER*4</b>	Input	System time from <b>time()</b> (standard version)
Return value	<b>character*24</b>	Output	System time as character string. Declare <b>ctime</b> and <i>string</i> as <b>character*24</b> .

The format of the **ctime** returned value is shown in the following example. It is described in the **man** page **ctime(3C)**.

Example: `ctime()`:

```
demo% cat tctime.f
      character*24 ctime, string
      INTEGER*4  n, time
      n = time()
      string = ctime( n )
      write(*,*) 'ctime: ', string
      end
demo% f95 tctime.f
demo% a.out
      ctime: Wed Dec  9 13:50:05 1998
demo%
```

### 1.4.51.3 `ltime`: Split System Time to Month, Day, ... (Local)

This routine dissects a system time into month, day, and so forth, for the local time zone.

The subroutine is called by:

---

`call ltime( stime, tarray )`

<i>stime</i>	<b>INTEGER*4</b>	Input	System time from <code>time()</code> (standard version)
<i>tarray</i>	<b>INTEGER*4(9)</b>	Output	System time, local, as day, month, year, ...

For the meaning of the elements in `tarray`, see the next section.

Example: `ltime()`:

```
demo% cat tltime.f
      integer*4  stime, tarray(9), time
      stime = time()
      call ltime( stime, tarray )
      write(*,*) 'ltime: ', tarray
      end
demo% f95 tltime.f
demo% a.out
      ltime: 25 49 10 12 7 91 1 223 1
demo%
```

### 1.4.51.4 `gmtime`: Split System Time to Month, Day, ... (GMT)

This routine dissects a system time into month, day, and so on, for GMT.

The subroutine is:

<b>call gmtime( stime, tarray )</b>			
<i>stime</i>	<b>INTEGER*4</b>	Input	System time from <b>time()</b> (standard version)
<i>tarray</i>	<b>INTEGER*4(9)</b>	Output	System time, GMT, as day, month, year, ...

Example: **gmtime**:

```
demo% cat tgmtime.f
      integer*4 stime, tarray(9), time
      stime = time()
      call gmtime( stime, tarray )
      write(*,*) 'gmtime: ', tarray
      end
demo% f95t tgmtime.f
demo% a.out
gmtime:  12  44  19  18  5  94  6  168  0
demo%
```

Here are the **tarray()** values for **ltime** and **gmtime**: index, units, and range:

1	Seconds (0 - 61)	6	Year - 1900
2	Minutes (0 - 59)	7	Day of week (Sunday = 0)
3	Hours (0 - 23)	8	Day of year (0 - 365)
4	Day of month (1 - 31)	9	Daylight Saving Time, 1 if DST in effect
5	Months since January (0 - 11)		

These values are defined by the C library routine **ctime(3C)**, which explains why the system may return a count of seconds *greater than* 59. See also: **idate(3F)**, and **fdate(3F)**.

### 1.4.51.5 **ctime64, gmtime64, ltime64: System Time Routines for 64-bit Environments**

These are versions of the corresponding routines **ctime**, **gmtime**, and **ltime**, to provide portability on 64-bit environments. They are identical to these routines except that the input variable *stime* must be **INTEGER\*8**.

When used in a 32-bit environment with an **INTEGER\*8** *stime*, if the value of *stime* is beyond the **INTEGER\*4** range **ctime64** returns all asterisks, while **gmtime** and **ltime** fill the *tarray* array with -1.



## 1.4.52 **ttynam, isatty: Get Name of a Terminal Port**

**ttynam** and **isatty** handle terminal port names.

### 1.4.52.1 **ttynam: Get Name of a Terminal Port**

The function **ttynam** returns a blank padded path name of the terminal device associated with logical unit *lunit*.

The function is called by:

---

**CHARACTER ttynam\*24**

*name* = **ttynam**( *lunit* )

<i>lunit</i>	<b>INTEGER*4</b>	Input	Logical unit
Return value	<b>character*n</b>	Output	If nonblank returned: <i>name</i> =path name of device on <i>lunit</i> . Size <i>n</i> must be large enough for the longest path name.  If empty string (all blanks) returned: <i>lunit</i> is not associated with a terminal device in the directory, <b>/dev</b>

### 1.4.52.2 **isatty: Is this Unit a Terminal?**

The function **isatty** returns true or false depending on whether or not logical unit *lunit* is a terminal device or not.

The function is called by:

---

*terminal* = **isatty**( *lunit* )

<i>lunit</i>	<b>INTEGER*4</b>	Input	Logical unit
Return value	<b>LOGICAL*4</b>	Output	<i>terminal</i> =true: It is a terminal device <i>terminal</i> =false: It is <i>not</i> a terminal device

Example: Determine if *lunit* is a **tty**:

```

character*12 name, ttynam
integer*4 lunit /5/
logical*4 isatty, terminal
terminal = isatty( lunit )
name = ttynam( lunit )
write(*,*) 'terminal = ', terminal, ', name = "', name, '"'
end

```

The output is:

```
terminal = T, name = "/dev/tty1 "
```

## 1.4.53 **unlink: Remove a File**

The function is called by:

---

<b>INTEGER*4 unlink</b>			
<i>n</i> = <b>unlink</b> ( <i>patnam</i> )			
<i>patnam</i>	<b>character*n</b>	Input	File name
Return value	<b>INTEGER*4</b>	Output	<i>n</i> =0: OK <i>n</i> >0: Error

---

The function **unlink** removes the file specified by path name *patnam*. If this is the last link to the file, the contents of the file are lost.

Example: **unlink()**—Remove the **tunlink.data** file:

```
demo% cat tunlink.f
      call unlink( 'tunlink.data' )
      end
demo% f95 tunlink.f
demo% ls tunl*
tunlink.f tunlink.data
demo% a.out
demo% ls tunl*
tunlink.f
```

See also: **unlink(2)**, **link(3F)**, and **perror(3F)**. Note: the path names cannot be longer than **MAXPATHLEN** as defined in **<sys/param.h>**.

## 1.4.54 **wait: Wait for a Process to Terminate**

The function is:

---

```
INTEGER*4 wait
n = wait( status )
```

---

<i>status</i>	<b>INTEGER*4</b>	Output	Termination status of the child process
Return value	<b>INTEGER*4</b>	Output	$n > 0$ : Process ID of the child process $n < 0$ : $n$ =System error code; see <b>wait(2)</b> .

**wait** suspends the caller until a signal is received, or one of its child processes terminates. If any child has terminated since the last **wait**, return is immediate. If there are no children, return is immediate with an error code.

Example: Code fragment using **wait()**:

```

INTEGER*4 n, status, wait
...
n = wait( status )
if ( n .lt. 0 ) stop 'wait: error'
...
end

```

See also: **wait(2)**, **signal(3F)**, **kill(3F)**, and **perorr(3F)**.



# Fortran 95 Intrinsic Functions

---

This chapter lists the intrinsic function names recognized by the **f95** compiler.

## 2.1 Standard Fortran 95 Generic Intrinsic Functions

The generic Fortran 95 intrinsic functions are grouped in this section by functionality as they appear in the Fortran 95 standard.

The arguments shown are the names that can be used as argument keywords when using the keyword form, as in `cmplx(Y=B, KIND=M, X=A)`.

Consult the Fortran 95 standard for the detailed specifications of these generic intrinsic procedures.

### 2.1.1 Argument Presence Inquiry Function

Generic Intrinsic Name	Description
<code>PRESENT</code>	Argument presence

### 2.1.2 Numeric Functions

Generic Intrinsic Name	Description
<code>ABS (A)</code>	Absolute value
<code>AIMAG (Z)</code>	Imaginary part of a complex number

Generic Intrinsic Name	Description
<b>AINT</b> (A [, KIND])	Truncation to whole number
<b>ANINT</b> (A [, KIND])	Nearest whole number
<b>CEILING</b> (A [, KIND])	Least integer greater than or equal to number
<b>CMPLX</b> (X [, Y, KIND])	Conversion to complex type
<b>CONJG</b> (Z)	Conjugate of a complex number
<b>DBLE</b> (A)	Conversion to double precision real type
<b>DIM</b> (X, Y)	Positive difference
<b>DPROD</b> (X, Y)	Double precision real product
<b>FLOOR</b> (A [, KIND])	Greatest integer less than or equal to number
<b>INT</b> (A [, KIND])	Conversion to integer type
<b>MAX</b> (A1, A2 [, A3, ...])	Maximum value
<b>MIN</b> (A1, A2 [, A3, ...])	Minimum value
<b>MOD</b> (A, P)	Remainder function
<b>MODULO</b> (A, P)	Modulo function
<b>NINT</b> (A [, KIND])	Nearest integer
<b>REAL</b> (A [, KIND])	Conversion to real type
<b>SIGN</b> (A, B)	Transfer of sign

## 2.1.3 Mathematical Functions

Generic Intrinsic Name	Description
<b>ACOS</b> (X)	Arccosine
<b>ASIN</b> (X)	Arcsine
<b>ATAN</b> (X)	Arctangent
<b>ATAN2</b> (Y, X)	Arctangent
<b>COS</b> (X)	Cosine
<b>COSH</b> (X)	Hyperbolic cosine
<b>EXP</b> (X)	Exponential

Generic Intrinsic Name	Description
<b>LOG (X)</b>	Natural logarithm
<b>LOG10 (X)</b>	Common logarithm (base 10)
<b>SIN (X)</b>	Sine
<b>SINH (X)</b>	Hyperbolic sine
<b>SQRT (X)</b>	Square root
<b>TAN (X)</b>	Tangent
<b>TANH (X)</b>	Hyperbolic tangent

## 2.1.4 Character Functions

Generic Intrinsic Name	Description
<b>ACHAR (I)</b>	Character in given position in ASCII collating sequence
<b>ADJUSTL (STRING)</b>	Adjust left
<b>ADJUSTR (STRING)</b>	Adjust right
<b>CHAR (I [, KIND])</b>	Character in given position in processor collating sequence
<b>IACHAR (C)</b>	Position of a character in ASCII collating sequence
<b>ICHAR (C)</b>	Position of a character in processor collating sequence
<b>INDEX (STRING, SUBSTRING [, BACK])</b>	Starting position of a substring
<b>LEN_TRIM (STRING)</b>	Length without trailing blank characters
<b>LGE (STRING_A, STRING_B)</b>	Lexically greater than or equal
<b>LGT (STRING_A, STRING_B)</b>	Lexically greater than
<b>LLE (STRING_A, STRING_B)</b>	Lexically less than or equal
<b>LLT (STRING_A, STRING_B)</b>	Lexically less than
<b>REPEAT (STRING, NCOPIES)</b>	Repeated concatenation
<b>SCAN (STRING, SET [, BACK])</b>	Scan a string for a character in a set
<b>TRIM (STRING)</b>	Remove trailing blank characters

Generic Intrinsic Name	Description
<b>VERIFY</b> (STRING, SET [, BACK])	Verify the set of characters in a string

## 2.1.5 Character Inquiry Function

Generic Intrinsic Name	Description
<b>LEN</b> (STRING)	Length of a character entity

## 2.1.6 Kind Functions

Generic Intrinsic Name	Description
<b>KIND</b> (X)	Kind type parameter value
<b>SELECTED_INT_KIND</b> (R)	Integer kind type parameter value, given range
<b>SELECTED_REAL_KIND</b> ([P, R])	Real kind type parameter value, given precision and range

## 2.1.7 Logical Function

Generic Intrinsic Name	Description
<b>LOGICAL</b> (L [, KIND])	Convert between objects of type logical with different kind type parameters

## 2.1.8 Numeric Inquiry Functions

Generic Intrinsic Name	Description
<b>DIGITS</b> (X)	Number of significant digits of the model
<b>EPSILON</b> (X)	Number that is almost negligible compared to one
<b>HUGE</b> (X)	Largest number of the model
<b>MAXEXPONENT</b> (X)	Maximum exponent of the model



Generic Intrinsic Name	Description
<b>MINEXPONENT (X)</b>	Minimum exponent of the model
<b>PRECISION (X)</b>	Decimal precision
<b>RADIX (X)</b>	Base of the model
<b>RANGE (X)</b>	Decimal exponent range
<b>TINY (X)</b>	Smallest positive number of the model

## 2.1.9 Bit Inquiry Function

Generic Intrinsic Name	Description
<b>BIT_SIZE (I)</b>	Number of bits of the model

## 2.1.10 Bit Manipulation Functions

Generic Intrinsic Name	Description
<b>BTEST (I, POS)</b>	Bit testing
<b>IAND (I, J)</b>	Logical AND
<b>IBCLR (I, POS)</b>	Clear bit
<b>IBITS (I, POS, LEN)</b>	Bit extraction
<b>IBSET (I, POS)</b>	Set bit
<b>IEOR (I, J)</b>	Exclusive OR
<b>IOR (I, J)</b>	Inclusive OR
<b>ISHFT (I, SHIFT)</b>	Logical shift
<b>ISHFTC (I, SHIFT [, SIZE])</b>	Circular shift
<b>NOT (I)</b>	Logical complement

## 2.1.11 Transfer Function

Generic Intrinsic Name	Description
<b>TRANSFER</b> (SOURCE, MOLD [, SIZE])	Treat first argument as if of type of second argument

## 2.1.12 Floating-Point Manipulation Functions

Generic Intrinsic Name	Description
<b>EXPONENT</b> (X)	Exponent part of a model number
<b>FRACTION</b> (X)	Fractional part of a number
<b>NEAREST</b> (X, S)	Nearest different processor number in given direction
<b>RRSPACING</b> (X)	Reciprocal of the relative spacing of model numbers near given number
<b>SCALE</b> (X, I)	Multiply a real by its base to an integer power
<b>SET_EXPONENT</b> (X, I)	Set exponent part of a number
<b>SPACING</b> (X)	Absolute spacing of model numbers near given number

## 2.1.13 Vector and Matrix Multiply Functions

Generic Intrinsic Name	Description
<b>DOT_PRODUCT</b> (VECTOR_A, VECTOR_B)	Dot product of two rank-one arrays
<b>MATMUL</b> (MATRIX_A, MATRIX_B)	Matrix multiplication

## 2.1.14 Array Reduction Functions

Generic Intrinsic Name	Description
<b>ALL</b> (MASK [, DIM])	True if all values are true
<b>ANY</b> (MASK [, DIM])	True if any value is true
<b>COUNT</b> (MASK [, DIM])	Number of true elements in an array

Generic Intrinsic Name	Description
<b>MAXVAL</b> (ARRAY, DIM [, MASK]) or <b>MAXVAL</b> (ARRAY [, MASK])	Maximum value in an array
<b>MINVAL</b> (ARRAY, DIM [, MASK]) or <b>MINVAL</b> (ARRAY [, MASK])	Minimum value in an array
<b>PRODUCT</b> (ARRAY, DIM [, MASK]) or <b>PRODUCT</b> (ARRAY [, MASK])	Product of array elements
<b>SUM</b> (ARRAY, DIM [, MASK]) or <b>SUM</b> (ARRAY [, MASK])	Sum of array elements

## 2.1.15 Array Inquiry Functions

Generic Intrinsic Name	Description
<b>ALLOCATED</b> (ARRAY)	Array allocation status
<b>LBOUND</b> (ARRAY [, DIM])	Lower dimension bounds of an array
<b>SHAPE</b> (SOURCE)	Shape of an array or scalar
<b>SIZE</b> (ARRAY [, DIM])	Total number of elements in an array
<b>UBOUND</b> (ARRAY [, DIM])	Upper dimension bounds of an array

## 2.1.16 Array Construction Functions

Generic Intrinsic Name	Description
<b>MERGE</b> (TSOURCE, FSOURCE, MASK)	Merge under mask
<b>PACK</b> (ARRAY, MASK [, VECTOR])	Pack an array into an array of rank one under a mask
<b>SPREAD</b> (SOURCE, DIM, NCOPIES)	Replicates array by adding a dimension
<b>UNPACK</b> (VECTOR, MASK, FIELD)	Unpack an array of rank one into an array under a mask

## 2.1.17 Array Reshape Function

Generic Intrinsic Name	Description
<b>RESHAPE</b> (SOURCE, SHAPE[, PAD, ORDER])	Reshape an array

## 2.1.18 Array Manipulation Functions

Generic Intrinsic Name	Description
<b>CSHIFT</b> (ARRAY, SHIFT [, DIM])	Circular shift
<b>EOSHIFT</b> (ARRAY, SHIFT [, BOUNDARY, DIM])	End-off shift
<b>TRANSPOSE</b> (MATRIX)	Transpose of an array of rank two

## 2.1.19 Array Location Functions

Generic Intrinsic Name	Description
<b>MAXLOC</b> (ARRAY, DIM [, MASK]) or <b>MAXLOC</b> (ARRAY [, MASK])	Location of a maximum value in an array
<b>MINLOC</b> (ARRAY, DIM [, MASK]) or <b>MINLOC</b> (ARRAY [, MASK])	Location of a minimum value in an array

## 2.1.20 Pointer Association Status Functions

Generic Intrinsic Name	Description
<b>ASSOCIATED</b> (POINTER [, TARGET])	Association status inquiry or comparison
<b>NULL</b> ([MOLD])	Returns disassociated pointer

## 2.1.21 System Environment Procedures

Generic Intrinsic Name	Description
<b>COMMAND_ARGUMENT_COUNT</b> ( )	Returns number of command arguments
<b>GET_COMMAND</b> ([ <b>COMMAND</b> , <b>LENGTH</b> , <b>STATUS</b> ])	Returns entire command that invoked the program
<b>GET_COMMAND_ARGUMENT</b> ( <b>NUMBER</b> [, <b>VALUE</b> , <b>LENGTH</b> , <b>STATUS</b> ])	Returns a command argument
<b>GET_ENVIRONMENT_VARIABLE</b> ( <b>NAME</b> [, <b>VALUE</b> , <b>LENGTH</b> , <b>STATUS</b> , <b>TRIM_NAME</b> ])	Obtain the value of an environment variable.

## 2.1.22 Intrinsic Subroutines

Generic Intrinsic Name	Description
<b>CPU_TIME</b> ( <b>TIME</b> )	Obtain processor time
<b>DATE_AND_TIME</b> ([ <b>DATE</b> , <b>TIME</b> , <b>ZONE</b> , <b>VALUES</b> ])	Obtain date and time
<b>MVBITS</b> ( <b>FROM</b> , <b>FROMPOS</b> , <b>LEN</b> , <b>TO</b> , <b>TOPOS</b> )	Copies bits from one integer to another
<b>RANDOM_NUMBER</b> ( <b>HARVEST</b> )	Returns pseudorandom number
<b>RANDOM_SEED</b> ([ <b>SIZE</b> , <b>PUT</b> , <b>GET</b> ])	Initializes or restarts the pseudorandom number generator
<b>SYSTEM_CLOCK</b> ([ <b>COUNT</b> , <b>COUNT_RATE</b> , <b>COUNT_MAX</b> ])	Obtain data from the system clock

## 2.1.23 Specific Names for Intrinsic Functions

TABLE 2-1 Specific and Generic Names for Fortran 95 Intrinsic Functions

	Specific Name	Generic Name	Argument Type
	ABS (A)	ABS (A)	default real
	ACOS (X)	ACOS (X)	default real
	AIMAG (Z)	AIMAG (Z)	default complex
	AINT (A)	AINT (A)	default real
	ALOG (X)	LOG (X)	default real
	ALOG10 (X)	LOG10 (X)	default real
#	AMAX0 (A1, A2 [, A3, ...])	REAL (MAX (A1, A2 [, A3, ...]))	default integer
#	AMAX1 (A1, A2 [, A3, ...])	MAX (A1, A2 [, A3, ...])	default real
#	AMIN0 (A1, A2 [, A3, ...])	REAL (MIN (A1, A2 [, A3, ...]))	default integer
#	AMIN1 (A1, A2 [, A3, ...])	MIN (A1, A2 [, A3, ...])	default real
	AMOD (A, P)	MOD (A, P)	default real
	ANINT (A)	ANINT (A)	default real
	ASIN (X)	ASIN (X)	default real
	ATAN (X)	ATAN (X)	default real
	ATAN2 (Y, X)	ATAN2 (Y, X)	default real
	CABS (A)	ABS (A)	default complex
	CCOS (X)	COS (X)	default complex
	CEXP (X)	EXP (X)	default complex
#	CHAR (I)	CHAR (I)	default integer
	CLOG (X)	LOG (X)	default complex
	CONJG (Z)	CONJG (Z)	default complex
	COS (X)	COS (X)	default real
	COSH (X)	COSH (X)	default real
	CSIN (X)	SIN (X)	default complex
	CSQRT (X)	SQRT (X)	default complex

TABLE 2-1 Specific and Generic Names for Fortran 95 Intrinsic Functions (Continued)

	Specific Name	Generic Name	Argument Type
	DABS (A)	ABS (A)	double precision
	DACOS (X)	ACOS (X)	double precision
	DASIN (X)	ASIN (X)	double precision
	DATAN (X)	ATAN (X)	double precision
	DATAN2 (Y, X)	ATAN2 (Y, X)	double precision
	DCOS (X)	COS (X)	double precision
	DCOSH (X)	COSH (X)	double precision
	DDIM (X, Y)	DIM (X, Y)	double precision
	DEXP (X)	EXP (X)	double precision
	DIM (X, Y)	DIM (X, Y)	default real
	DINT (A)	AINT (A)	double precision
	DLOG (X)	LOG (X)	double precision
	DLOG10 (X)	LOG10 (X)	double precision
#	DMAX1 (A1, A2 [, A3, ...])	MAX (A1, A2 [, A3, ...])	double precision
#	DMIN1 (A1, A2 [, A3, ...])	MIN (A1, A2 [, A3, ...])	double precision
	DMOD (A, P)	MOD (A, P)	double precision
	DNINT (A)	ANINT (A)	double precision
	DPROD (X, Y)	DPROD (X, Y)	default real
	DSIGN (A, B)	SIGN (A, B)	double precision
	DSIN (X)	SIN (X)	double precision
	DSINH (X)	SINH (X)	double precision
	DSQRT (X)	SQRT (X)	double precision
	DTAN (X)	TAN (X)	double precision
	DTANH (X)	TANH (X)	double precision
	EXP (X)	EXP (X)	default real
#	FLOAT (A)	REAL (A)	default integer
	IABS (A)	ABS (A)	default integer
#	ICHAR (C)	ICHAR (C)	default character

TABLE 2-1 Specific and Generic Names for Fortran 95 Intrinsic Functions (Continued)

	Specific Name	Generic Name	Argument Type
	IDIM (X, Y)	DIM (X, Y)	default integer
#	IDINT (A)	INT (A)	double precision
	IDNINT (A)	NINT (A)	double precision
#	IFIX (A)	INT (A)	default real
	INDEX (STRING, SUBSTRING)	INDEX (STRING, SUBSTRING)	default character
#	INT (A)	INT (A)	default real
	ISIGN (A, B)	SIGN (A, B)	default integer
	LEN (STRING)	LEN (STRING)	default character
#	LGE (STRING_A, STRING_B)	LGE (STRING_A, STRING_B)	default character
#	LGT (STRING_A, STRING_B)	LGT (STRING_A, STRING_B)	default character
#	LLE (STRING_A, STRING_B)	LLE (STRING_A, STRING_B)	default character
#	LLT (STRING_A, STRING_B)	LLT (STRING_A, STRING_B)	default character
#	MAX0 (A1, A2 [, A3, ...])	MAX (A1, A2 [, A3, ...])	default integer
#	MAX1 (A1, A2 [, A3, ...])	INT (MAX (A1, A2 [, A3, ...]))	default real
#	MIN0 (A1, A2 [, A3, ...])	MIN (A1, A2 [, A3, ...])	default integer
#	MIN1 (A1, A2 [, A3, ...])	INT (MIN (A1, A2 [, A3, ...]))	default real
	MOD (A, P)	MOD (A, P)	default integer
	NINT (A)	NINT (A)	default real
#	REAL (A)	REAL (A)	default integer
	SIGN (A, B)	SIGN (A, B)	default real
	SIN (X)	SIN (X)	default real
	SINH (X)	SINH (X)	default real
#	SNGL (A)	REAL (A)	double precision
	SQRT (X)	SQRT (X)	default real
	TAN (X)	TAN (X)	default real
	TANH (X)	TANH (X)	default real

Functions marked with # cannot be used as an actual argument. “double precision” means double-precision real.



## 2.2 Fortran 2003 Module Routines

The Fortran 2003 standard provides a set of intrinsic modules that define features to support IEEE arithmetic and interoperability with the C language.

### 2.2.1 IEEE Arithmetic and Exceptions Modules

The Fortran 2003 standard intrinsic modules **IEEE\_EXCEPTIONS**, **IEEE\_ARITHMETIC**, and **IEEE\_FEATURES** to support new features in the proposed language standard to support IEEE arithmetic and IEEE exception handling.

The draft standard defines a set of inquiry functions, elemental functions, kind functions, elemental subroutines, and nonelemental subroutines. These are listed in the tables that follow.

To access these functions and subroutines, the calling routine must include

```
USE, INTRINSIC :: IEEE_ARITHMETIC, IEEE_EXCEPTIONS
```

See the Fortran standard (<http://www.j3-fortran.org>) for details.

#### 2.2.1.1 Inquiry Functions

The module **IEEE\_EXCEPTIONS** contains the following inquiry functions.

Function	Description
<b>IEEE_SUPPORT_FLAG</b> (FLAG[,X])	Inquire whether the processor supports an exception.
<b>IEEE_SUPPORT_HALTING</b> (FLAG)	Inquire whether the processor supports control of halting after an exception.

The module **IEEE\_ARITHMETIC** contains the following inquiry functions.

Function	Description
<b>IEEE_SUPPORT_DATATYPE</b> ([X])	Inquire whether the processor supports IEEE arithmetic.
<b>IEEE_SUPPORT_DENORMAL</b> ([X])	Inquire whether the processor supports denormalized numbers.
<b>IEEE_SUPPORT_DIVIDE</b> ([X])	Inquire whether the processor supports divide with the accuracy specified by the IEEE standard.
<b>IEEE_SUPPORT_INF</b> ([X])	Inquire whether the processor supports the IEEE infinity.

<b>IEEE_SUPPORT_IO</b> ( [X] )	Inquire whether the processor supports IEEE base conversion rounding during formatted input/output.
<b>IEEE_SUPPORT_NAN</b> ( [X] )	Inquire whether the processor supports the IEEE Not-a-Number.
<b>IEEE_SUPPORT_ROUNDING</b> ( VAL[ , X] )	Inquire whether the processor supports a particular rounding mode.
<b>IEEE_SUPPORT_SQRT</b> ( [X] )	Inquire whether the processor supports the IEEE square root.
<b>IEEE_SUPPORT_STANDARD</b> ( [X] )	Inquire whether the processor supports all IEEE facilities.

### 2.2.1.2 Elemental Functions

The module **IEEE\_ARITHMETIC** contains the following elemental functions for real **X** and **Y** for which **IEEE\_SUPPORT\_DATATYPE(X)** and **IEEE\_SUPPORT\_DATATYPE(Y)** are true.

Function	Description
<b>IEEE_CLASS</b> (X)	IEEE class.
<b>IEEE_COPY_SIGN</b> (X,Y)	IEEE copysign function.
<b>IEEE_IS_FINITE</b> (X)	Determine if value is finite.
<b>IEEE_IS_NAN</b> (X)	Determine if value is IEEE Not-a-Number
<b>IEEE_IS_NORMAL</b> (X)	Determine if a value is normal.
<b>IEEE_IS_NEGATIVE</b> (X)	Determine if value is negative.
<b>IEEE_LOGB</b> (X)	Unbiased exponent in the IEEE floating point format.
<b>IEEE_NEXT_AFTER</b> (X,Y)	Returns the next representable neighbor of <b>X</b> in the direction toward <b>Y</b> .
<b>IEEE_REM</b> (X,Y)	The IEEE REM remainder function, $X - Y*N$ where <b>N</b> is the integer nearest to the exact value of <b>X/Y</b> .
<b>IEEE_RINT</b> (X)	Round to an integer value according to the current rounding mode.
<b>IEEE_SCALB</b> (X,I)	Returns $X*2^{**I}$
<b>IEEE_UNORDERED</b> (X,Y)	IEEE unordered function. True if <b>X</b> or <b>Y</b> is a NaN and false otherwise.
<b>IEEE_VALUE</b> (X,CLASS)	Generate an IEEE value.

### 2.2.1.3 Kind Function

The module **IEEE\_ARITHMETIC** contains the following transformational function:

Function	Description
<b>IEEE_SELECTED_REAL_KIND</b> ( [P, ] [R] )	Kind type parameter value fo an IEEE real with given precision and range.

### 2.2.1.4 Elemental Subroutines

The module **IEEE\_EXCEPTIONS** contains the following elemental subroutines.

Subroutine	Description
<b>IEEE_GET_FLAG</b> (FLAG, FLAG_VALUE)	Get an exception flag.
<b>IEEE_GET_HALTING_MODE</b> (FLAG, HALTING)	GEt halting mode for an exception.

### 2.2.1.5 Nonelemental Subroutines

The module **IEEE\_EXCEPTIONS** contains the following nonelemental subroutines.

Subroutine	Description
<b>IEEE_GET_STATUS</b> (STATUS_VALUE)	Get the current state of the floating point environment.
<b>IEEE_SET_FLAG</b> (FLAG, FLAG_VALUE)	Set an exception flag.
<b>IEEE_SET_HALTING_MODE</b> (FLAG, HALTING)	Controls continuation or halting on exceptions.
<b>IEEE_SET_STATUS</b> (STATUS_VALUE)	Restore the state of the floating point environment.

The module **IEEE\_ARITHMETIC** contains the following nonelemental subroutines.

Subroutine	Description
<b>IEEE_GET_ROUNDING_MODE</b> (ROUND_VAL)	Get the current IEEE rounding mode.
<b>IEEE_SET_ROUNDING_MODE</b> (ROUND_VAL)	Set the current IEEE rounding mode.

## 2.2.2 C Binding Module

The Fortran 2003 standard provides a means of referencing C language procedures. The **ISO\_C\_BINDING** module defines three support procedures as intrinsic module functions. Accessing these functions requires

```
USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_LOC, C_PTR, C_ASSOCIATED
```

in the calling routine. The procedures defined in the module are

Function	Description
<b>C_LOC(X)</b>	Returns the C address of the argument
<b>C_ASSOCIATED(C_PTR_1 [, C_PTR_2])</b>	Indicates the association status of <b>C_PTR_1</b> or indicates whether <b>C_PTR_1</b> and <b>C_PTR_2</b> are associated with the same entity.
<b>C_F_POINTER(CPTR, FPTR [, SHAPE])</b>	Associates a pointer with the target of a C pointer and specifies its shape.

For details on the **ISO\_C\_BINDING** intrinsic module, see Chapter 15 of the Fortran 2003 standard at <http://www.j3-fortran.org>.

## 2.3 Non-Standard Fortran 95 Intrinsic Functions

The following functions are considered intrinsics by the **f95** compiler, but are not part of the Fortran 95 standard.

### 2.3.1 Basic Linear Algebra Functions (BLAS)

When compiling with **-xknown\_lib=blas**, the compiler will recognize calls to the following routines as intrinsics and will optimize for and link to the Sun Performance Library implementation. The compiler will ignore user-supplied versions of these routines.

TABLE 2-2 BLAS Intronics

Function	Description
CAXPY DAXPY SAXPY ZAXPY	Product of a scalar and a vector plus a vector
CCOPY DCOPY SCOPY ZCOPY	Copy a vector
CDOTC CDOTU DDOT SDOT ZDOTC ZDOTU	Dot product (inner product)
CSCAL DSCAL SSCAL ZSCAL	Scale a vector

See the *Sun Performance Library User's Guide* for more information on these routines.

## 2.3.2 Interval Arithmetic Intrinsic Functions

The following table lists intrinsic functions that are recognized by the compiler when compiling for interval arithmetic (`-xia`). For details, see the *Fortran 95 Interval Arithmetic Programming Reference*.

DINTERVAL	DIVIX	INF	INTERVAL
IEMPTY	MAG	MID	MIG
NDIGITS	QINTERVAL	SINTERVAL	SUP

VDABS	VDACOS	VDASIN	VDATAN
VDATAN2	VDCEILING	VDCOS	VDCOSH
VDEXP	VDFLOOR	VDINF	VDINT
VDISEMPTY	VDLOG	VDLOG10	VDMAG
VDMID	VDMIG	VDMOD	VDNINT
VDSIGN	VDSIN	VDSINH	VDSQRT
VDSUP	VDTAN	VDTANH	VDWID
VQABS	VQCEILING	VQFLOOR	VQINF
VQINT	VQISEMPTY	VQMAG	VQMID
VQMIG	VQNINT	VQSUP	VQWID
VSABS	VSACOS	VSASIN	VSATAN
VSATAN2	VSCEILING	VSCOS	VSCOSH
VSEXP	VSFLOOR	VSINF	VSINT
VSISEMPTY	VSLOG	VSLOG10	VSMAG
VSMID	VSMIG	VSMOD	VSNINT
VSSIGN	VSSIN	VSSINH	VSSQRT
VSSUP	VSTAN	VSTANH	VSVID
WID			

## 2.3.3 Other Vendor Intrinsic Functions

The **f95** compiler recognizes a variety of legacy intrinsic functions that were defined by Fortran compilers from other vendors, including Cray Research, Inc. These are obsolete and their use should be avoided.

TABLE 2-3 Intrinsic Functions From Cray CF90 and Other Compilers

Function	Arguments	Description
<b>CLOC</b>	(([C]=]c)	Obtains the address of a character object
<b>COMPL</b>	(([I=]i)	Bit-by-bit complement of a word. Use NOT(i) instead
<b>COT</b>	(([X=]x)	Generic cotangent. (Also: DCOT, QCOT)
<b>CSMG</b>	(([I=]i,[J=]j,[K=]k)	Conditional Scalar Merge

TABLE 2-3 Intrinsic Functions From Cray CF90 and Other Compilers (Continued)

Function	Arguments	Description
<b>DSHIFTL</b>	(([I=]i,[J=]j,[K=]k)	Double-object left shift of i and j by k bits
<b>DSHIFTR</b>	(([I=]i,[J=]j,[K=]k)	Double-object right shift of i and j by k bits
<b>EQV</b>	(([I=]i,[J=]j)	Logical equivalence. Use IOER(i,j) instead.
<b>FCD</b>	(([I=]i,[J=]j)	Constructs a character pointer
<b>GETPOS</b>	(([I=]i)	Obtains file position
<b>IBCHNG</b>	(([I=]i, [POS=]j)	Generic function to change specified bit in a word.
<b>ISHA</b>	(([I=]i, [SHIFT=]j)	Generic arithmetic shift
<b>ISHC</b>	(([I=]i, [SHIFT=]j)	Generic circular shift
<b>ISHL</b>	(([I=]i, [SHIFT=]j)	Generic left shift
<b>LEADZ</b>	(([I=]i)	Counts number of leading 0 bits
<b>LENGTH</b>	(([I=]i)	Returns the number of Cray words successfully transferred
<b>LOC</b>	(([I=]i)	Returns the address of a variable (See “1.4.32 <b>loc</b> : Return the Address of an Object” on page 60)
<b>NEQV</b>	(([I=]i,[J=]j)	Logical non-equivalence. Use IOER(i,j) instead.
<b>POPCNT</b>	(([I=]i)	Counts number of bits set to 1
<b>POPPAR</b>	(([I=]i)	Computes bit population parity
<b>SHIFT</b>	(([I=]i,[J=]j)	Shift left circular. Use ISHFT(i,j) or ISHFTC(i,j,k) instead.
<b>SHIFTA</b>	(([I=]i,[J=]j)	Arithmetic shift with sign extension.
<b>SHIFTL</b>	(([I=]i,[J=]j)	Shift left with zero fill. Use ISHFT(i,j) or ISHFTC(i,j,k) instead.
<b>SHIFTR</b>	(([I=]i,[J=]j)	Shift right with zero fill. Use ISHFT(i,j) or ISHFTC(i,j,k) instead.
<b>TIMEF</b>	()	Returns elapsed time since the first call
<b>UNIT</b>	(([I=]i)	Returns status of BUFFERIN or BUFFEROUT
<b>XOR</b>	(([I=]i,[J=]j)	Logical exclusive OR. Use IOER(i,j) instead.

See also “2.3.4.2 Memory Functions” on page 112 for a list of VMS Fortran 77 intrinsics.

## 2.3.4 Other Extensions

The Fortran 95 compiler recognizes the following additional intrinsic functions:

### 2.3.4.1 MPI\_SIZEOF

---

**MPI\_SIZEOF** (*x*, *size*, *error*)

Returns the size in bytes of the machine representation of the given variable, *x*. If *x* is an array, it returns the size of the base element and not the size of the whole array

---

<i>x</i>	input; variable or array of arbitrary type
<i>size</i>	output; integer; size in bytes of <i>x</i>
<i>error</i>	output; integer; set to an error code if an error detected, zero otherwise

---

### 2.3.4.2 Memory Functions

Memory allocation, reallocation, and deallocation functions **malloc()**, **realloc()**, and **free()** are implemented as **f95** intrinsics. See “[1.4.35 malloc, malloc64, realloc, free: Allocate/Reallocate/Deallocate Memory](#)” on page 63 for details.



## FORTRAN 77 and VMS Intrinsic Functions

---

This chapter lists the set of FORTRAN 77 intrinsic functions accepted by **f95** and is provided to aid migration of programs from legacy FORTRAN 77 to Fortran 95.

**f95** recognizes as intrinsic functions all the FORTRAN 77 and VMS functions listed in this chapter, along with all the Fortran 95 functions listed in the previous chapter. As an aid to migrating legacy FORTRAN 77 programs to **f95**, compiling with **-f77=intrinsics** causes the compiler to recognize only FORTRAN 77 and VMS functions as intrinsics, and not the Fortran 95 intrinsics.

Intrinsic functions that are Sun extensions of the ANSI FORTRAN 77 standard are marked with  $\boxtimes$ . Programs using non-standard intrinsics and library functions may not be portable to other platforms.

Intrinsic functions have *generic* and *specific* names when they accept arguments of more than one data type. In general, the *generic* name returns a value with the same data type as its argument. However, there are exceptions such as the type conversion functions (Table 3-2) and the inquiry functions (Table 3-7). The function may also be called by one of its *specific* names to handle a specific argument data type.

With functions that work on more than one data item (for example, **sign(a1, a2)**), all the data arguments must be the same type.

In the following tables, the FORTRAN 77 intrinsic functions are listed by:

- Intrinsic Function— description of what the function does
- Definition— a mathematical definition
- No. of Args.— number of arguments the function accepts
- Generic Name— the function's generic name
- Specific Names— the function's specific names
- Argument Type— data type associated with each specific name
- Function Type— data type returned for specific argument data type

---

**Note** – Compiler option **-xtypemap** changes the default sizes of variables and has an affect on intrinsic references. See “3.4 Remarks” on page 127, and the discussion of default sizes and alignment in the *Fortran User’s Guide*.

---

## 3.1 Arithmetic and Mathematical Functions

This section details arithmetic, type conversion, trigonometric, and other functions. “a” stands for a function’s single argument, “a1” and “a2” for the first and second arguments of a two argument function, and “ar” and “ai” for the real and imaginary parts of a function’s complex argument.

### 3.1.1 Arithmetic Functions

TABLE 3-1 Fortran 77 Arithmetic Functions

Intrinsic Function	Definition	No. of Args.	Generic Name	Specific Names	Argument Type	Function Type
Absolute value <i>See Note (6).</i>	$ a  = (ar^2 + ai^2)^{1/2}$	1	<b>ABS</b>	<b>IABS</b> <b>ABS</b> <b>DABS</b> <b>CABS</b> <b>QABS</b> ☒ <b>ZABS</b> ☒ <b>CDABS</b> ☒ <b>CQABS</b> ☒	<b>INTEGER</b> <b>REAL</b> <b>DOUBLE</b> <b>COMPLEX</b> <b>REAL*16</b> <b>DOUBLE COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>	<b>INTEGER</b> <b>REAL</b> <b>DOUBLE</b> <b>REAL</b> <b>REAL*16</b> <b>DOUBLE</b> <b>DOUBLE</b> <b>REAL*16</b>
Truncation <i>See Note (1).</i>	$\text{int}(a)$	1	<b>AINT</b>	<b>AINT</b> <b>DINT</b> <b>QINT</b> ☒	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>
Nearest whole number	$\text{int}(a+.5)$ if $a \geq 0$ $\text{int}(a-.5)$ if $a < 0$	1	<b>ANINT</b>	<b>ANINT</b> <b>DNINT</b> <b>QNINT</b> ☒	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>

TABLE 3-1 Fortran 77 Arithmetic Functions (Continued)

Intrinsic Function	Definition	No. of Args.	Generic Name	Specific Names	Argument Type	Function Type
Nearest integer	$\text{int}(a+.5)$ if $a \geq 0$ $\text{int}(a-.5)$ if $a < 0$	1	<b>NINT</b>	<b>NINT</b>	<b>REAL</b>	<b>INTEGER</b>
				<b>IDNINT</b>	<b>DOUBLE</b>	<b>INTEGER</b>
				<b>IQNINT</b> ☐	<b>REAL*16</b>	<b>INTEGER</b>
Remainder <i>See Note (1).</i>	$a1 - \text{int}(a1/a2) * a2$	2	<b>MOD</b>	<b>MOD</b>	<b>INTEGER</b>	<b>INTEGER</b>
				<b>AMOD</b>	<b>REAL</b>	<b>REAL</b>
				<b>DMOD</b>	<b>DOUBLE</b>	<b>DOUBLE</b>
				<b>QMOD</b> ☐	<b>REAL*16</b>	<b>REAL*16</b>
Transfer of sign	$ a1 $ if $a2 \geq 0$ $- a1 $ if $a2 < 0$	2	<b>SIGN</b>	<b>ISIGN</b>	<b>INTEGER</b>	<b>INTEGER</b>
				<b>SIGN</b>	<b>REAL</b>	<b>REAL</b>
				<b>DSIGN</b>	<b>DOUBLE</b>	<b>DOUBLE</b>
				<b>QSIGN</b> ☐	<b>REAL*16</b>	<b>REAL*16</b>
Positive difference	$a1 - a2$ if $a1 > a2$ $0$ if $a1 \leq a2$	2	<b>DIM</b>	<b>IDIM</b>	<b>INTEGER</b>	<b>INTEGER</b>
				<b>DIM</b>	<b>REAL</b>	<b>REAL</b>
				<b>DDIM</b>	<b>DOUBLE</b>	<b>DOUBLE</b>
				<b>QDIM</b> ☐	<b>REAL*16</b>	<b>REAL*16</b>
Double and quad products	$a1 * a2$	2	-	<b>DPROD</b>	<b>REAL</b>	<b>DOUBLE</b>
				<b>QPROD</b> ☐	<b>DOUBLE</b>	<b>REAL*16</b>
Choosing largest value	$\text{max}(a1, a2, \dots)$	$\geq 2$	<b>MAX</b>	<b>MAX0</b>	<b>INTEGER</b>	<b>INTEGER</b>
				<b>AMAX1</b>	<b>REAL</b>	<b>REAL</b>
				<b>DMAX1</b>	<b>DOUBLE</b>	<b>DOUBLE</b>
				<b>QMAX1</b> ☐	<b>REAL*16</b>	<b>REAL*16</b>
			<b>AMAX0</b>	<b>INTEGER</b>	<b>REAL</b>	
			<b>MAX1</b>	<b>MAX1</b>	<b>REAL</b>	<b>INTEGER</b>

TABLE 3-1 Fortran 77 Arithmetic Functions (Continued)

Intrinsic Function	Definition	No. of Args.	Generic Name	Specific Names	Argument Type	Function Type
Choosing smallest value	min(a1, a2, ...)	≥2	MIN	MIN0	INTEGER	INTEGER
				AMIN1	REAL	REAL
				DMIN1	DOUBLE	DOUBLE
				QMIN1 ☒	REAL*16	REAL*16
			AMIN0	AMIN0	INTEGER	REAL
			MIN1	MIN1	REAL	INTEGER

## 3.1.2 Type Conversion Functions

TABLE 3-2 Fortran 77 Type Conversion Functions

Conversion to	No. of Args	Generic Name	Specific Names	Argument Type	Function Type
INTEGER <i>See Note (1).</i>	1	INT	-	INTEGER	INTEGER
			INT	REAL	INTEGER
			IFIX	REAL	INTEGER
			IDINT	DOUBLE	INTEGER
			-	COMPLEX	INTEGER
			-	COMPLEX*16	INTEGER
			-	COMPLEX*32	INTEGER
			IQINT ☒	REAL*16	INTEGER

TABLE 3-2 Fortran 77 Type Conversion Functions (Continued)

Conversion to	No. of Args	Generic Name	Specific Names	Argument Type	Function Type
REAL <i>See Note (2).</i>	1	REAL	REAL	INTEGER	REAL
			FLOAT	INTEGER	REAL
			-	REAL	REAL
			SNGL	DOUBLE	REAL
			SNGLQ □	REAL*16	REAL
			-	COMPLEX	REAL
			-	COMPLEX*16	REAL
			-	COMPLEX*32	REAL
		FLOATK	INTEGER*8	REAL*4	
DOUBLE <i>See Note (3).</i>	1	DBLE	DBLE	INTEGER	DOUBLE PRECISION
			DFLOAT	INTEGER	DOUBLE PRECISION
			DFLOATK	INTEGER*8	DOUBLE PRECISION
			DREAL □	REAL	DOUBLE PRECISION
			-	DOUBLE	DOUBLE PRECISION
			-	COMPLEX	DOUBLE PRECISION
			-	COMPLEX*16	DOUBLE PRECISION
			-	REAL*16	DOUBLE PRECISION
			-	COMPLEX*32	DOUBLE PRECISION
			-	REAL*16	DOUBLE PRECISION
DBLEQ □-	COMPLEX*32	DOUBLE PRECISION			

TABLE 3-2 Fortran 77 Type Conversion Functions (Continued)

Conversion to	No. of Args	Generic Name	Specific Names	Argument Type	Function Type
REAL*16 <i>See Note (3).</i>	1	QREAL <sup>□</sup> QEXT <sup>□</sup>	QREAL <sup>□</sup> QFLOAT <sup>□</sup> - QEXT <sup>□</sup> QEXTD <sup>□</sup> - - -	INTEGER INTEGER REAL INTEGER DOUBLE REAL*16 COMPLEX COMPLEX*16 COMPLEX*32	REAL*16 REAL*16 REAL*16 REAL*16 REAL*16 REAL*16 REAL*16 REAL*16 REAL*16
COMPLEX <i>See Notes (4) and (8).</i>	1 or 2	CMPLX	- - - - - -	INTEGER REAL DOUBLE REAL*16 COMPLEX COMPLEX*16 COMPLEX*32	COMPLEX COMPLEX COMPLEX COMPLEX COMPLEX COMPLEX COMPLEX
DOUBLE COMPLEX <i>See Note (8).</i>	1 or 2	DCMPLX@	- - - - - -	INTEGER REAL DOUBLE REAL*16 COMPLEX COMPLEX*16 COMPLEX*32	DOUBLE COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX

TABLE 3-2 Fortran 77 Type Conversion Functions (Continued)

Conversion to	No. of Args	Generic Name	Specific Names	Argument Type	Function Type
COMPLEX*32 <i>See Note (8).</i>	1 or 2	QCMLX@	- - - - - - -	INTEGER REAL DOUBLE REAL*16 COMPLEX COMPLEX*16 COMPLEX*32	COMPLEX*32 COMPLEX*32 COMPLEX*32 COMPLEX*32 COMPLEX*32 COMPLEX*32 COMPLEX*32
INTEGER <i>See Note (5).</i>	1	- -	ICHAR IACHAR □	CHARACTER	INTEGER
CHARACTER <i>See Note (5).</i>	1	- -	CHAR ACHAR □	INTEGER	CHARACTER

On an ASCII platform, including Sun systems:

- **ACHAR** is a nonstandard synonym for **CHAR**
- **IACHAR** is a nonstandard synonym for **ICHAR**

On a non-ASCII platform, **ACHAR** and **IACHAR** were intended to provide a way to deal directly with ASCII.

## 3.1.3 Trigonometric Functions

TABLE 3-3 Fortran 77 Trigonometric Functions

Intrinsic Function	Definition	Args	Generic Name	Specific Names	Argument Type	Function Type
Sine <i>See Note (7).</i>	$\sin(a)$	1	<b>SIN</b>	<b>SIN</b> <b>DSIN</b> <b>QSIN</b> ☐ <b>CSIN</b> <b>ZSIN</b> ☐ <b>CDSIN</b> ☐ <b>CQSIN</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b> <b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b> <b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>
Sine (degrees) <i>See Note (7).</i>	$\sin(a)$	1	<b>SIND</b> ☐	<b>SIND</b> ☐ <b>DSIND</b> ☐ <b>QSIND</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>
Cosine <i>See Note (7).</i>	$\cos(a)$	1	<b>COS</b>	<b>COS</b> <b>DCOS</b> <b>QCOS</b> ☐ <b>CCOS</b> <b>ZCOS</b> ☐ <b>CDCOS</b> ☐ <b>CQCOS</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b> <b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b> <b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>
Cosine (degrees) <i>See Note (7).</i>	$\cos(a)$	1	<b>COSD</b> ☐	<b>COSD</b> ☐ <b>DCOSD</b> ☐ <b>QCOSD</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>
Tangent <i>See Note (7).</i>	$\tan(a)$	1	<b>TAN</b>	<b>TAN</b> <b>DTAN</b> <b>QTAN</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>
Tangent (degrees) <i>See Note (7).</i>	$\tan(a)$	1	<b>TAND</b> ☐	<b>TAND</b> ☐ <b>DTAND</b> ☐ <b>QTAND</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>



TABLE 3-3 Fortran 77 Trigonometric Functions (Continued)

Intrinsic Function	Definition	Args	Generic Name	Specific Names	Argument Type	Function Type
Arcsine <i>See Note (7).</i>	arcsin(a)	1	ASIN	ASIN	REAL	REAL
				DASIN	DOUBLE	DOUBLE
				QASIN ☐	REAL*16	REAL*16
Arcsine (degrees) <i>See Note (7).</i>	arcsin(a)	1	ASIND ☐	ASIND ☐	REAL	REAL
				DASIND ☐	DOUBLE	DOUBLE
				QASIND ☐	REAL*16	REAL*16
Arccosine <i>See Note (7).</i>	arccos(a)	1	ACOS	ACOS	REAL	REAL
				DACOS	DOUBLE	DOUBLE
				QACOS ☐	REAL*16	REAL*16
Arccosine (degrees) <i>See Note (7).</i>	arccos(a)	1	ACOSD ☐	ACOSD ☐	REAL	REAL
				DACOSD ☐	DOUBLE	DOUBLE
				QACOSD ☐	REAL*16	REAL*16
Arctangent <i>See Note (7).</i>	arctan(a)	1	ATAN	ATAN	REAL	REAL
				DATAN	DOUBLE	DOUBLE
				QATAN ☐	REAL*16	REAL*16
	arctan (a1/a2)	2	ATAN2	ATAN2	REAL	REAL
				DATAN2	DOUBLE	DOUBLE
				QATAN2 ☐	REAL*16	REAL*16
Arctangent (degrees) <i>See Note (7).</i>	arctan(a)	1	ATAND ☐	ATAND ☐	REAL	REAL
				DATAND ☐	DOUBLE	DOUBLE
				QATAND ☐	REAL*16	REAL*16
	arctan (a1/a2)	2	ATAN2D ☐	ATAN2D ☐	REAL	REAL
				DATAN2D ☐	DOUBLE	DOUBLE
				QATAN2D ☐	REAL*16	REAL*16
Hyperbolic Sine <i>See Note (7).</i>	sinh(a)	1	SINH	SINH	REAL	REAL
				DSINH	DOUBLE	DOUBLE
				QSINH ☐	REAL*16	REAL*16

TABLE 3-3 Fortran 77 Trigonometric Functions (Continued)

Intrinsic Function	Definition	Args	Generic Name	Specific Names	Argument Type	Function Type
Hyperbolic Cosine <i>See Note (7).</i>	$\cosh(a)$	1	<b>COSH</b>	<b>COSH</b> <b>DCOSH</b> <b>QCOSH</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>
Hyperbolic Tangent <i>See Note (7).</i>	$\tanh(a)$	1	<b>TANH</b>	<b>TANH</b> <b>DTANH</b> <b>QTANH</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>

### 3.1.4 Other Mathematical Functions

TABLE 3-4 Other Fortran 77 Mathematical Functions

Intrinsic Function	Definition	No. of Args.	Generic Name	Specific Names	Argument Type	Function Type
Imaginary part of a complex number <i>See Note (6).</i>	$ai$	1	<b>IMAG</b>	<b>AIMAG</b> <b>DIMAG</b> ☐ <b>QIMAG</b> ☐	<b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b>
Conjugate of a complex number <i>See Note (6).</i>	$(ar, -ai)$	1	<b>CONJG</b>	<b>CONJG</b> <b>DCONJG</b> ☐ <b>QCONJG</b> ☐	<b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>	<b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>
Square root	$a^{1/2}$	1	<b>SQRT</b>	<b>SQRT</b> <b>DSQRT</b> <b>QSQRT</b> ☐ <b>CSQRT</b> <b>ZSQRT</b> ☐ <b>CDSQRT</b> ☐ <b>CQSQRT</b> ☐	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b> <b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>	<b>REAL</b> <b>DOUBLE</b> <b>REAL*16</b> <b>COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>DOUBLE COMPLEX</b> <b>COMPLEX*32</b>

TABLE 3-4 Other Fortran 77 Mathematical Functions (Continued)

Intrinsic Function	Definition	No. of Args.	Generic Name	Specific Names	Argument Type	Function Type
Cube root <i>See Note(8')</i> .	$a^{**}(1/3)$	1	<b>CBRT</b>	CBRT ☐ DCBRT ☐ QCBRT ☐ CCBRT ☐ ZCBRT ☐ CDCBRT ☐ CQCBRT ☐	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32
Exponential	$e^{**}a$	1	<b>EXP</b>	EXP DEXP QEXP ☐ CEXP ZEXP ☐ CDEXP ☐ CQEXP ☐	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32
Natural logarithm	$\log(a)$	1	<b>LOG</b>	ALOG DLOG QLOG ☐ CLOG ZLOG ☐ CDLOG ☐ CQLOG ☐	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32
Common logarithm	$\log_{10}(a)$	1	<b>LOG10</b>	ALOG10 DLOG10 QLOG10 ☐	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
Error function( <i>See note below</i> )	$\text{erf}(a)$	1	<b>ERF</b>	ERF ☐ DERF ☐	REAL DOUBLE	REAL DOUBLE
Error function	$1.0 - \text{erf}(a)$	1	<b>ERFC</b>	ERFC ☐ DERFC ☐	REAL DOUBLE	REAL DOUBLE

- The error function:  $2/\text{sqrt}(\pi) \times \text{integral from } 0 \text{ to } a \text{ of } \exp(-t^2t) dt$

## 3.2 Character Functions

TABLE 3-5 Fortran 77 Character Functions

Intrinsic Function	Definition	No. of Args.	Specific Names	Argument Type	Function Type
Conversion <i>See Note (5).</i>	Conversion to character	1	<b>CHAR</b>	<b>INTEGER</b>	<b>CHARACTER</b>
	Conversion to integer <i>See also:</i> <a href="#">Table 3-2.</a>	1	<b>ACHAR</b> □ <b>ICHAR</b> <b>IACHAR</b> □	<b>CHARACTER</b>	<b>INTEGER</b>
Index of a substring	Location of substring a2 in string a1 <i>See Note (10).</i>	2	<b>INDEX</b>	<b>CHARACTER</b>	<b>INTEGER</b>
Length	Length of character entity <i>See Note (11).</i>	1	<b>LEN</b>	<b>CHARACTER</b>	<b>INTEGER</b>
Lexically greater than or equal	$a1 \geq a2$ <i>See Note (12).</i>	2	<b>LGE</b>	<b>CHARACTER</b>	<b>LOGICAL</b>
Lexically greater than	$a1 > a2$ <i>See Note (12).</i>	2	<b>LGT</b>	<b>CHARACTER</b>	<b>LOGICAL</b>
Lexically less than or equal	$a1 \leq a2$ <i>See Note (12).</i>	2	<b>LLE</b>	<b>CHARACTER</b>	<b>LOGICAL</b>
Lexically less than	$a1 < a2$ <i>See Note (12).</i>	2	<b>LLT</b>	<b>CHARACTER</b>	<b>LOGICAL</b>

On an ASCII platform (including Sun systems):

- **ACHAR** is a nonstandard synonym for **CHAR**
- **IACHAR** is a nonstandard synonym for **ICHAR**

On a non-ASCII platform, **ACHAR** and **IACHAR** were intended to provide a way to deal directly with ASCII.

## 3.3 Miscellaneous Functions

Other miscellaneous functions include bitwise functions, environmental inquiry functions, and memory allocation and deallocation functions.

### 3.3.1 Bit Manipulation $\alpha$

None of these functions are part of the FORTRAN 77 Standard.

TABLE 3-6 Fortran 77 Bitwise Functions

Bitwise Operations	No. of Args.	Specific Name	Argument Type	Function Type
Complement	1	<b>NOT</b>	<b>INTEGER</b>	<b>INTEGER</b>
And	22	<b>AND IAND</b>	<b>INTEGER</b>	<b>INTEGER</b>
Inclusive or	22	<b>OR IOR</b>	<b>INTEGER</b>	<b>INTEGER</b>
Exclusive or	22	<b>XOR IEOR</b>	<b>INTEGER</b>	<b>INTEGER</b>
Shift <i>See Note (14).</i>	2	<b>ISHFT</b>	<b>INTEGER</b>	<b>INTEGER</b>
Left shift <i>See Note (14).</i>	2	<b>LSHIFT</b>	<b>INTEGER</b>	<b>INTEGER</b>
Right shift <i>See Note (14).</i>	2	<b>RSHIFT</b>	<b>INTEGER</b>	<b>INTEGER</b>
Logical right shift <i>See Note (14).</i>	2	<b>LRSHT</b>	<b>INTEGER</b>	<b>INTEGER</b>
Circular shift	3	<b>ISHFTC</b>	<b>INTEGER</b>	<b>INTEGER</b>
Bit extraction	3	<b>IBITS</b>	<b>INTEGER</b>	<b>INTEGER</b>
Bit set	2	<b>IBSET</b>	<b>INTEGER</b>	<b>INTEGER</b>
Bit test	2	<b>BTEST</b>	<b>INTEGER</b>	<b>LOGICAL</b>
Bit clear	2	<b>IBCLR</b>	<b>INTEGER</b>	<b>INTEGER</b>

The above functions are available as intrinsic or extrinsic functions. See also the discussion of the library bit manipulation routines in the *Fortran Library Reference* manual.

### 3.3.2 Environmental Inquiry Functions $\alpha$

None of these functions are part of the FORTRAN 77 Standard.

TABLE 3-7 Fortran 77 Environmental Inquiry Functions

Definition	No. of Args.	Generic Name	Argument Type	Function Type
Base of Number System	1	EPBASE	INTEGER	INTEGER
			REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
Number of Significant Bits	1	EPPREC	INTEGER	INTEGER
			REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
Minimum Exponent	1	EPEMIN	REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
Maximum Exponent	1	EPEMAX	REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
Least Nonzero Number	1	EPTINY	REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16
Largest Number Representable	1	EPHUGE	INTEGER	INTEGER
			REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16
Epsilon <i>See Note (16).</i>	1	EPMRSP	REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16

### 3.3.3 Memory $\alpha$

None of these functions are part of the FORTRAN 77 Standard.

TABLE 3-8 Fortran 77 Memory Functions

Intrinsic Function	Definition	No. of Args	Specific Name	Argument Type	Function Type
Location	Address of <i>See Note (17).</i>	1	<b>LOC</b>	<i>Any</i>	<b>INTEGER*4INTEGER*</b>
Allocate	Allocate memory and return address. <i>See Note (17).</i>	1	<b>MALLOC</b> <b>MALLOC64</b>	<b>INTEGER*4</b> <b>INTEGER*8</b>	<b>INTEGER</b> <b>INTEGER*8</b>
Deallocate	Deallocate memory allocated by <b>MALLOC</b> . <i>See Note (17).</i>	1	<b>FREE</b>	<i>Any</i>	-
Size	Return the size of the argument in bytes. <i>See Note (18).</i>	1	<b>SIZEOF</b>	<i>Any expression</i>	<b>INTEGER</b>

## 3.4 Remarks

The following remarks apply to all of the intrinsic function tables in this chapter.

- The abbreviation **DOUBLE** stands for **DOUBLE PRECISION**.
- An intrinsic that takes **INTEGER** arguments accepts **INTEGER\*2**, **INTEGER\*4**, or **INTEGER\*8**.
- **INTEGER** intrinsics that take **INTEGER** arguments return values of **INTEGER** type determined as follows. Note that the **-xtypemap** option may alter the default sizes of actual arguments:
  - **mod sign dim max min and iand or ior xor ieor**— size of the value returned is the largest of the sizes of the arguments.
  - **abs ishft lshift rshift lrshft ibset ivclr ishftc ibits**— size of the value returned is the size of the first argument.
  - **int eabase epprec**— size of the value returned is the size of default **INTEGER**.
  - **ephuge**— size of the value returned is the size of the default **INTEGER**, or the size of the argument, whichever is largest.

Options that change the default data sizes also change the way some intrinsics are used. For example, with **-dbl** in effect, a call to **ZCOS** with a **DOUBLE COMPLEX** argument will automatically become a call to **CQCOS** because the argument has been promoted to **COMPLEX\*32**. The following functions have this capability:

**aimag alog amod cabs ccbprt ccos cdabs cdbprt cdcos cdexp cdlog cdsin cdsqrt  
cexp clog csin csqrt dabs dacos dacosd dasin dasind datan datand dcbprt dconjg**

**dcos dcosd dcosh ddim derf derfc dexp dimag dint dlog dmod dnint dprod dsign dsin dsind dsinh dsqrt dtan dtand dtanh idnint iidnnt jidnnt zabs zcbrt zcos zexp zlog zsin zsqrt**

- The following functions permit arguments of an integer or logical type of any size:  
**and iand ieor iand ieor iior inot ior jand jieor jior jnot lrshft lshift not or rshift xor**
- An intrinsic that is shown to return a *default* **REAL**, **DOUBLE PRECISION**, **COMPLEX**, or **DOUBLE COMPLEX** value will return the prevailing type depending on certain compilation options. For example, if compiled with **-xtypemap=real:64,double:64**:
  - A call to a **REAL** function returns **REAL\*8**
  - A call to a **DOUBLE PRECISION** function returns **REAL\*8**
  - A call to a **COMPLEX** function returns **COMPLEX\*16**
  - A call to a **DOUBLE COMPLEX** function returns **COMPLEX\*16**

Other options that alter the data sizes of default data types are **-r8** and **-dbl**, which also promote **DOUBLE** to **QUAD**. The **-xtypemap=** option provides more flexibility than these older compiler options and is preferred.

A function with a generic name returns a value with the same type as the argument—except for type conversion functions, the nearest integer function, the absolute value of a complex argument, and others. If there is more than one argument, they must all be of the same type.

- If a function name is used as an actual argument, then it must be a specific name.
- If a function name is used as a dummy argument, then it does not identify an intrinsic function in the subprogram, and it has a data type according to the same rules as for variables and arrays.

## 3.4.1 Notes on Functions

Tables and notes 1 through 12 are based on the “Table of Intrinsic Functions,” from *ANSI X3.9-1978 Programming Language FORTRAN*, with the Fortran extensions added.

### (1) **INT**

If **A** is type integer, then **INT(A)** is **A**.

If **A** is type real or double precision, then:

if  $|A| < 1$ , then **INT(A)** is 0  
if  $|A| \geq 1$ , then **INT(A)** is the greatest integer that does not exceed the magnitude of **A**, and whose sign is the same as the sign of **A**. (Such a mathematical integer value may be too large to fit in the computer integer type.)

If **A** is type complex or double complex, then apply the above rule to the real part of **A**.

If **A** is type real, then **IFIX(A)** is the same as **INT(A)**.



**(2) REAL**

If **A** is type real, then **REAL (A)** is **A**.

If **A** is type integer or double precision, then **REAL (A)** is as much precision of the significant part of **A** as a real datum can contain.

If **A** is type complex, then **REAL (A)** is the real part of **A**.

If **A** is type double complex, then **REAL (A)** is as much precision of the significant part of the real part of **A** as a real datum can contain.

**(3) DBLE**

If **A** is type double precision, then **DBLE (A)** is **A**.

If **A** is type integer or real, then **DBLE (A)** is as much precision of the significant part of **A** as a double precision datum can contain.

If **A** is type complex, then **DBLE (A)** is as much precision of the significant part of the real part of **A** as a double precision datum can contain.

If **A** is type **COMPLEX\*16**, then **DBLE (A)** is the real part of **A**.

**(3') QREAL**

If **A** is type **REAL\*16**, then **QREAL (A)** is **A**.

If **A** is type integer, real, or double precision, then **QREAL (A)** is as much precision of the significant part of **A** as a **REAL\*16** datum can contain.

If **A** is type complex or double complex, then **QREAL (A)** is as much precision of the significant part of the real part of **A** as a **REAL\*16** datum can contain.

If **A** is type **COMPLEX\*16** or **COMPLEX\*32**, then **QREAL (A)** is the real part of **A**.

**(4) CMLPX**

If **A** is type complex, then **CMLPX(A)** is **A**.

If **A** is type integer, real, or double precision, then **CMLPX(A)** is **REAL (A) + 0i**.

If **A1** and **A2** are type integer, real, or double precision, then **CMLPX(A1, A2)** is **REAL (A1) + REAL (A2) \* i**.

If **A** is type double complex, then **CMLPX(A)** is **REAL ( DBLE(A) ) + i\*REAL( DIMAG(A) )**.

If **CMLPX** has two arguments, then they must be of the same type, and they may be one of integer, real, or double precision.

If **CMPLX** has one argument, then it may be one of integer, real, double precision, complex, **COMPLEX\*16**, or **COMPLEX\*32**.

(4') **DCMPLX**

If **A** is type **COMPLEX\*16**, then **DCMPLX(A)** is **A**.

If **A** is type integer, real, or double precision, then **DCMPLX(A)** is **DBLE(A) + 0i**.

If **A1** and **A2** are type integer, real, or double precision, then **DCMPLX(A1, A2)** is **DBLE(A1) + DBLE(A2)\*i**.

If **DCMPLX** has two arguments, then they must be of the same type, and they may be one of integer, real, or double precision.

If **DCMPLX** has one argument, then it may be one of integer, real, double precision, complex, **COMPLEX\*16**, or **COMPLEX\*32**.

(5) **ICHAR**

**ICHAR(A)** is the position of **A** in the collating sequence.

The first position is 0, the last is **N-1**,  $0 \leq \text{ICHAR}(A) \leq N-1$ , where **N** is the number of characters in the collating sequence, and **A** is of type character of length one.

**CHAR** and **ICHAR** are inverses in the following sense:

- **ICHAR(CHAR(I)) = I**, for  $0 \leq I \leq N-1$
- **CHAR(ICHAR(C)) = C**, for any character **C** capable of representation in the processor

(6) **COMPLEX**

A **COMPLEX** value is expressed as an ordered pair of reals, (**ar**, **ai**), where **ar** is the real part, and **ai** is the imaginary part.

(7) Radians

All angles are expressed in radians, unless the “Intrinsic Function” column includes the “degrees” remark.

(8) **COMPLEX** Function

The result of a function of type **COMPLEX** is the principal value.

(8') **CBRT**

If **a** is of **COMPLEX** type, **CBRT** results in **COMPLEX RT1=(A, B)**, where:  $A \geq 0.0$ , and  $-60 \text{ degrees} \leq \arctan(B/A) < +60 \text{ degrees}$ .

Other two possible results can be evaluated as follows:

- **RT2 = RT1 \* (-0.5, square\_root(0.75))**
- **RT3 = RT1 \* (-0.5, square\_root(0.75))**

## (9) Argument types

All arguments in an intrinsic function reference must be of the same type.

## (10) INDEX

**INDEX**(*X*, *Y*) is the place in *X* where *Y* starts. That is, it is the starting position within character string *X* of the first occurrence of character string *Y*.

If *Y* does not occur in *X*, then **INDEX**(*X*, *Y*) is 0.

If **LEN**(*X*) < **LEN**(*Y*), then **INDEX**(*X*, *Y*) is 0.

**INDEX** returns default **INTEGER\*4** data. If compiling for a 64-bit environment, the compiler will issue a warning if the result overflows the **INTEGER\*4** data range. To use **INDEX** in a 64-bit environment with character strings larger than the **INTEGER\*4** limit (2 Gbytes), the **INDEX** function and the variables receiving the result must be declared **INTEGER\*8**.

## (11) LEN

**LEN** returns the declared length of the **CHARACTER** argument variable. The actual value of the argument is of no importance.

**LEN** returns default **INTEGER\*4** data. If compiling for a 64-bit environment, the compiler will issue a warning if the result overflows the **INTEGER\*4** data range. To use **LEN** in a 64-bit environment with character variables larger than the **INTEGER\*4** limit (2 Gbytes), the **LEN** function and the variables receiving the result must be declared **INTEGER\*8**.

## (12) Lexical Compare

**LGE**( *X*, *Y* ) is true if *X*=*Y*, or if *X* follows *Y* in the collating sequence; otherwise, it is false.

**LGT**( *X*, *Y* ) is true if *X* follows *Y* in the collating sequence; otherwise, it is false.

**LLE**( *X*, *Y* ) is true if *X*=*Y*, or if *X* precedes *Y* in the collating sequence; otherwise, it is false.

**LLT**( *X*, *Y* ) is true if *X* precedes *Y* in the collating sequence; otherwise, it is false.

If the operands for **LGE**, **LGT**, **LLE**, and **LLT** are of unequal length, the shorter operand is considered as if it were extended on the right with blanks.

## (13) Bit Functions

There are other bitwise operations in VMS Fortran, but these are not implemented.

## (14) Shift

**LSHIFT** shifts *a1* logically *left* by *a2* bits (inline code).

**LRSHFT** shifts *a1* logically *right* by *a2* bits (inline code).

**RSHIFT** shifts *a1* arithmetically *right* by *a2* bits.

**ISHFT** shifts *a1* logically *left* if *a2* > 0 and *right* if *a2* < 0.

The **LSHIFT** and **RSHIFT** functions are the Fortran analogs of the C << and >> operators. As in C, the semantics depend on the hardware.

The behavior of the shift functions with an out of range shift count is hardware dependent and generally unpredictable. In this release, shift counts larger than 31 result in hardware dependent behavior.

(15) Environmental inquiries

Only the type of the argument is significant.

(16) Epsilon

Epsilon is the least  $e$ , such that  $1.0 + e \neq 1.0$ .

(17) **LOC**, **MALLOC**, and **FREE**

The **LOC** function returns the address of a variable or of an external procedure. The function call **MALLOC** (  $n$  ) allocates a block of at least  $n$  bytes, and returns the address of that block.

**LOC** returns default **INTEGER\*4** in 32-bit environments, **INTEGER\*8** in 64-bit environments.

**MALLOC** is a library function and not an intrinsic in FORTRAN 77. It too returns default **INTEGER\*4** in 32-bit environments, **INTEGER\*8** in 64-bit environments. However, **MALLOC** must be explicitly declared **INTEGER\*8** when compiling for 64-bit environments.

The value returned by **LOC** or **MALLOC** should be stored in variables typed **POINTER**, **INTEGER\*4**, or **INTEGER\*8** in 64-bit environments. The argument to **FREE** must be the value returned by a previous call to **MALLOC** and hence should have data type **POINTER**, **INTEGER\*4**, or **INTEGER\*8**.

**MALLOC64** always takes an **INTEGER\*8** argument (size of memory request in bytes) and always returns an **INTEGER\*8** value. Use this routine rather than **MALLOC** when compiling programs that must run in both 32-bit and 64-bit environments. The receiving variable must be declared either **POINTER** or **INTEGER\*8**.

(18) **SIZEOF**

The **SIZEOF** intrinsic cannot be applied to arrays of an assumed size, characters of a length that is passed, or subroutine calls or names. **SIZEOF** returns default **INTEGER\*4** data. If compiling for a 64-bit environment, the compiler will issue a warning if the result overflows the **INTEGER\*4** data range. To use **SIZEOF** in a 64-bit environment with arrays larger than the **INTEGER\*4** limit (2 Gbytes), the **SIZEOF** function and the variables receiving the result must be declared **INTEGER\*8**.

## 3.5 VMS Intrinsic Functions

This section lists VMS FORTRAN intrinsic routines recognized by **f95**. They are, of course, nonstandard. ☐

## 3.5.1 VMS Double-Precision Complex

TABLE 3-9 VMS Double-Precision Complex Functions

Generic Name	Specific Names	Function	Argument Type	Result Type
	<b>CDABS</b>	Absolute value	<b>COMPLEX*16</b>	<b>REAL*8</b>
	<b>CDEXP</b>	Exponential, $e^{**a}$	<b>COMPLEX*16</b>	<b>COMPLEX*16</b>
	<b>CDLOG</b>	Natural log	<b>COMPLEX*16</b>	<b>COMPLEX*16</b>
	<b>CDSQRT</b>	Square root	<b>COMPLEX*16</b>	<b>COMPLEX*16</b>
	<b>CDSIN</b>	Sine	<b>COMPLEX*16</b>	<b>COMPLEX*16</b>
	<b>CDCOS</b>	Cosine	<b>COMPLEX*16</b>	<b>COMPLEX*16</b>
<b>DCMPLX</b>	<b>DCONJG</b>	Convert to <b>DOUBLE COMPLEX</b>	Any numeric	<b>COMPLEX*16</b>
	<b>DIMAG</b>	Complex conjugate	<b>COMPLEX*16</b>	<b>COMPLEX*16</b>
	<b>DREAL</b>	Imaginary part of complex	<b>COMPLEX*16</b>	<b>REAL*8</b>
		Real part of complex	<b>COMPLEX*16</b>	<b>REAL*8</b>

## 3.5.2 VMS Degree-Based Trigonometric

TABLE 3-10 VMS Degree-Based Trigonometric Functions

Generic Name	Specific Names	Function	Argument Type	Result Type
<b>SIND</b>	<b>SIND</b>	Sine	-	-
	<b>DSIND</b>		<b>REAL*4</b>	<b>REAL*4</b>
	<b>QSIND</b>		<b>REAL*8</b>	<b>REAL*8</b>
			<b>REAL*16</b>	<b>REAL*16</b>
<b>COSD</b>	<b>COSD</b>	Cosine	-	-
	<b>DCOSD</b>		<b>REAL*4</b>	<b>REAL*4</b>
	<b>QCOSD</b>		<b>REAL*8</b>	<b>REAL*8</b>
			<b>REAL*16</b>	<b>REAL*16</b>
<b>TAND</b>	<b>TAND</b>	Tangent	-	-
	<b>DTAND</b>		<b>REAL*4</b>	<b>REAL*4</b>
	<b>QTAND</b>		<b>REAL*8</b>	<b>REAL*8</b>
			<b>REAL*16</b>	<b>REAL*16</b>

TABLE 3-10 VMS Degree-Based Trigonometric Functions (Continued)

Generic Name	Specific Names	Function	Argument Type	Result Type
ASIND	ASIND	Arc sine	-	-
	DASIND		REAL*4	REAL*4
	QASIND		REAL*8	REAL*8
			REAL*16	REAL*16
ACOSD	ACOSD	Arc cosine	-	-
	DACOSD		REAL*4	REAL*4
	QACOSD		REAL*8	REAL*8
			REAL*16	REAL*16
ATAND	ATAND	Arc tangent	-	-
	DATAND		REAL*4	REAL*4
	QATAND		REAL*8	REAL*8
			REAL*16	REAL*16
ATAN2D	ATAN2D	Arc tangent of a1/a2	-	-
	DATAN2D		REAL*4	REAL*4
	QATAN2D		REAL*8	REAL*8
			REAL*16	REAL*16

### 3.5.3 VMS Bit-Manipulation

TABLE 3-11 VMS Bit-Manipulation Functions

Generic Name	Specific Names	Function	Argument Type	Result Type
IBITS	IIBITS	From <b>a1</b> , initial bit <b>a2</b> , extract <b>a3</b> bits	-	-
	JIBITS		INTEGER*2	INTEGER*2
	KIBITS		INTEGER*4	INTEGER*4
			INTEGER*8	INTEGER*8
ISHFT	IISHFT	Shift <b>a1</b> logically by <b>a2</b> bits; if <b>a2</b> positive shift left, if <b>a2</b> negative shift right	-	-
	JISHFT		INTEGER*2	INTEGER*2
	KISHFT		INTEGER*4	INTEGER*4
			INTEGER*8	INTEGER*8

TABLE 3-11 VMS Bit-Manipulation Functions (Continued)

Generic Name	Specific Names	Function	Argument Type	Result Type
ISHFTC	IISHFTC	In <b>a1</b> , circular shift by <b>a2</b> places, of right <b>a3</b> bits	-	-
	JISHFTC		INTEGER*2	INTEGER*2
			INTEGER*4	INTEGER*4
IAND	IIAND	Bitwise <b>AND</b> of <b>a1</b> , <b>a2</b>	-	-
	JIAND		INTEGER*2	INTEGER*2
			INTEGER*4	INTEGER*4
IOR	IIOR	Bitwise <b>OR</b> of <b>a1</b> , <b>a2</b>	-	-
	JIOR		INTEGER*2	INTEGER*2
	KIOR		INTEGER*4	INTEGER*4
			INTEGER*8	INTEGER*8
IEOR	IIEOR	Bitwise exclusive <b>OR</b> of <b>a1</b> , <b>a2</b>	-	-
	JIEOR		INTEGER*2	INTEGER*2
	KIEOR		INTEGER*4	INTEGER*4
			INTEGER*8	INTEGER*8
NOT	INOT	Bitwise complement	-	-
	JNOT		INTEGER*2	INTEGER*2
	KNOT		INTEGER*4	INTEGER*4
			INTEGER*8	INTEGER*8
IBSET	IIBSET	In <b>a1</b> , set bit <b>a2</b> to 1; return new <b>a1</b>	-	-
	JIBSET		INTEGER*2	INTEGER*2
	KIBSET		INTEGER*4	INTEGER*4
			INTEGER*8	INTEGER*8
BTEST	BITEST	If bit <b>a2</b> of <b>a1</b> is 1, return <b>.TRUE.</b>	-	-
	BJTEST		INTEGER*2	LOGICAL
	BKTEST		INTEGER*4	LOGICAL
			INTEGER*8	LOGICAL

TABLE 3-11 VMS Bit-Manipulation Functions (Continued)

Generic Name	Specific Names	Function	Argument Type	Result Type
<b>IBCLR</b>	<b>IIBCLR</b>	In <b>a1</b> , set bit <b>a2</b> to 0; return new <b>a1</b>	-	-
	<b>JIBCLR</b>		<b>INTEGER*2</b>	<b>INTEGER*2</b>
	<b>KIBCLR</b>		<b>INTEGER*4</b>	<b>INTEGER*4</b>
			<b>INTEGER*8</b>	<b>INTEGER*8</b>

## 3.5.4 VMS Multiple Integer Types

The possibility of multiple integer types is not addressed by the Fortran Standard. The compiler copes with their existence by treating a specific **INTEGER**-to-**INTEGER** function name (**IABS**, and so forth) as a special sort of generic. The argument type is used to select the appropriate runtime routine name, which is not accessible to the programmer.

VMS Fortran takes a similar approach, but makes the specific names available.

TABLE 3-12 VMS Integer Functions

Specific Names	Function	Argument Type	Result Type
<b>IIABS</b>	Absolute value	<b>INTEGER*2</b>	<b>INTEGER*2</b>
<b>JIABS</b>		<b>INTEGER*4</b>	<b>INTEGER*4</b>
<b>KIABS</b>		<b>INTEGER*8</b>	<b>INTEGER*8</b>
<b>IMAX0</b>	Maximum	<b>INTEGER*2</b>	<b>INTEGER*2</b>
<b>JMAX0</b>		<b>INTEGER*4</b>	<b>INTEGER*4</b>
<b>IMIN0</b>	Minimum	<b>INTEGER*2</b>	<b>INTEGER*2</b>
<b>JMIN0</b>		<b>INTEGER*4</b>	<b>INTEGER*4</b>
<b>IIDIM</b>	Positive difference	<b>INTEGER*2</b>	<b>INTEGER*2</b>
<b>JIDIM</b>		<b>INTEGER*4</b>	<b>INTEGER*4</b>
<b>KIDIM</b>		<b>INTEGER*8</b>	<b>INTEGER*8</b>
<b>IMOD</b>	Remainder of a1/a2	<b>INTEGER*2</b>	<b>INTEGER*2</b>
<b>JMOD</b>		<b>INTEGER*4</b>	<b>INTEGER*4</b>
<b>IISIGN</b>	Transfer of sign, $ a1  * \text{sign}(a2)$	<b>INTEGER*2</b>	<b>INTEGER*2</b>
<b>JISIGN</b>		<b>INTEGER*4</b>	<b>INTEGER*4</b>
<b>KISIGN</b>		<b>INTEGER*8</b>	<b>INTEGER*8</b>



# Index

---

## Numbers and Symbols

$(e^{**x})-1$ , 17, 20

## A

**abort**, 25

**access**, 25-26

accessible documentation, 10-11

**alarm**, 26-27

**and**, 27

arc

  cosh, 16, 19

  cosine, 16

  sine, 16

  sinh, 16

  tangent, 16

  tanh, 19

arguments, command line, **getarg**, 42-43

array functions

  Fortran 95 intrinsics, 98-99, 100

## B

Bessel functions, 17, 18, 20, 22

**bic**, 27

**bis**, 27

**bit**, 27

bit

  functions, 27

  move bits, **mvbits**, 67

bit manipulation functions

  FORTRAN 77 intrinsics, 125

  Fortran 95 intrinsics, 97-98

  VMS intrinsics, 134-136

bitwise

  and, 27

  complement, 27

  exclusive or, 27

  inclusive or, 27

BLAS (Basic Linear Algebra Subroutines), 108

## C

C binding functions, 108

ceiling, 16

change

  default directory, **chdir**, 30

  mode of a file, **chmod**, 30-31

character

  get a character **getc**, **fgetc**, 43

  put a character, **putc**, **fputc**, 69

character functions

  FORTRAN 77 intrinsics, 124

  Fortran 95 intrinsics, 95-96

**chdir**, 30

**chmod**, 30-31

command-line argument, **getarg**, 42

conversion by **long**, **short**, 61-62

conversion functions, FORTRAN 77

  intrinsics, 116-119

**ctime**, convert system time to character, 85, 86-87

**ctime64**, 88  
 cube root, 16  
 current working directory, **getcwd**, 45

## D

**d\_acos(x)**, 19,20  
**d\_acosd(x)**, 19,20  
**d\_acosh(x)**, 19,20  
**d\_acosp(x)**, 19,20  
**d\_acospi(x)**, 19,20  
**d\_addran()**, 21  
**d\_addrans()**, 21  
**d\_asin(x)**, 19  
**d\_asind(x)**, 19  
**d\_asinh(x)**, 19  
**d\_asinp(x)**, 19  
**d\_asinpi(x)**, 19  
**d\_atan(x)**, 19  
**d\_atan2(x)**, 20  
**d\_atan2d(x)**, 20  
**d\_atan2pi(x)**, 20  
**d\_atand(x)**, 19  
**d\_atanh(x)**, 19  
**d\_atanp(x)**, 19  
**d\_atanpi(x)**, 19  
**d\_cbrt(x)**, 20  
**d\_ceil(x)**, 20  
**d\_erf(x)**, 20  
**d\_erfc(x)**, 20  
**d\_expml(x)**, 20  
**d\_floor(x)**, 20  
**d\_hypot(x)**, 20  
**d\_infinity()**, 20  
**d\_j0(x)**, 20  
**d\_j1(x)**, 20  
**d\_jn(n, x)**, 20  
**d\_lcran()**, 21  
**d\_lcrans()**, 21  
**d\_lgamma(x)**, 21  
**d\_log1p(x)**, 21  
**d\_log2(x)**, 21  
**d\_logb(x)**, 21  
**d\_max\_normal()**, 22

**d\_max\_subnormal()**, 22  
**d\_min\_normal()**, 22  
**d\_min\_subnormal()**, 22  
**d\_nextafter(x,y)**, 22  
**d\_quiet\_nan(n)**, 22  
**d\_remainder(x,y)**, 22  
**d\_rint(x)**, 22  
**d\_scalbn(x,n)**, 22  
**d\_shufrens()**, 21  
**d\_signaling\_nan(n)**, 22  
**d\_significand(x)**, 22  
**d\_sin(x)**, 22  
**d\_sincos(x,s,c)**, 22  
**d\_sincosd(x,s,c)**, 22  
**d\_sincosp(x,s,c)**, 22  
**d\_sincospi(x,s,c)**, 22  
**d\_sind(x)**, 22  
**d\_sinh(x)**, 22  
**d\_sinp(x)**, 22  
**d\_sinpi(x)**, 22  
**d\_tan(x)**, 22  
**d\_tand(x)**, 22  
**d\_tanh(x)**, 22  
**d\_tanp(x)**, 22  
**d\_tanpi(x)**, 22  
**d\_y0(x)**, *bessel*, 22  
**d\_y1(x)**, *bessel*, 22  
**d\_yn(n,x)**, 22  
 data types, 13  
 date  
   and time, as characters, **fdate**, 36-37  
   current date, **date**, 31-33  
   **date\_and\_time**, 32-33  
**date\_and\_time**, 32-33  
 deallocate memory by **free**, 66  
 delay execution, **alarm**, 26-27  
 descriptor, get file, **getfd**, 46  
 directory  
   default change, **chdir**, 30  
   get current working directory, **getcwd**, 45  
 documentation, accessing, 9-11, 11  
 documentation index, 9  
 double-precision **libm** functions, 19-23  
**drand**, 73

**mtime**, 34-36

## E

elapsed time, 34-36

environment variables, **getenv**, 45-46

error

function, 16

handlers, I/O, 76

messages,  **perror**,  **gerror**,  **ierrno**, 67

errors and interrupts,  **longjmp**, 62

**etime**, 34-36

exception handling, 51, 55

execute an OS command,  **system**, 76, 77, 79, 84

execution time, 34-36

existence of file,  **access**, 25-26

**exit**, 36

## F

**fdate**, 36-37

**fgetc**, 44

file

descriptor, get,  **getfd**, 46

get file pointer,  **getfilep**, 46

mode,  **access**, 25-26

permissions,  **access**, 25-26

remove,  **unlink**, 90

rename, 74-75

status,  **stat**, 81-84

status,  **stat64**, 84

find substring,  **index**, 56

floating-point

IEEE definitions, 54

IEEE exception handling, 50

floating-point functions, Fortran 95 intrinsics, 98

**floatingpoint.h** header file, 54

floor, 17

**flush**, 37

**fork**, 38

Fortran 2003 module routines, 105-108

FORTTRAN 77, intrinsic functions, 113

Fortran 95

non-standard intrinsic functions, 108

standard generic intrinsic functions, 93-104

**fputc**, 69

**free**, 66

**fseek**, 38-40

**fseeko64**, 40-42

**fstat**, 81-84

**fstat64**, 84

**ftell**, 38-40

**ftello64**, 40-42

## G

**gerror**, 67

get

character  **getc**,  **fgetc**, 43

current working directory,  **getcwd**, 45

environment variables,  **getenv**, 45-46

file descriptor,  **getfd**, 46

file pointer,  **getfilep**, 46

group id,  **getgid**, 49

login name,  **getlog**, 48

process id,  **getpid**, 48

user id,  **getuid**, 48-49

**get\_io\_err\_handler**, 76-79

**getarg**, 42

**getc**, 43

**getcwd**, 45

**getenv**, 45-46

**getfd**, 46

**getfilep**, 46

**getgid**, 49

**getlog**, 48

**getpid**, 48

**getuid**, 48-49

**gmtime**, 86

**gmtime**, GMT, 88

**gmtime64**, 88

Greenwich Mean Time,  **gmtime**, 86

group ID, get,  **getgid**, 49

**H**

host name, get, **hostnm**, 49-50  
**hostnm**, 49-50  
hyperbolic cos, 16  
hyperbolic tan, 18, 22  
hypotenuse, 17

**I**

I/O error handlers, 76  
**iargc**, 42  
id, process, get, **getpid**, 48  
**id\_finite(x)**, 21  
**id\_fp\_class(x)**, 21  
**id\_rint(x)**, 21  
**id\_isinf(x)**, 21  
**id\_isnan(x)**, 21  
**id\_isnormal(x)**, 21  
**id\_issubnormal(x)**, 21  
**id\_iszero(x)**, 21  
**id\_logb(x)**, 21  
**id\_signbit(x)**, 21  
IEEE arithmetic, 50-55  
IEEE arithmetic and exceptions (Fortran 2003), 105-108  
IEEE environment, 54-55  
    exception handling, 55  
    rounding mode, 54  
**ieee\_flags**, 50-55  
**ieee\_handler**, 50-55  
**ierrno**, 67  
**IMPLICIT**, 13  
**index**, 55-57  
**inmax**, 57  
inode, 81  
inquiry functions  
    FORTRAN 77 intrinsics, 125-126  
    Fortran 95 intrinsics, 93, 96, 99  
integer, conversion by **long**, **short**, 61-62  
interrupts and errors, **longjmp**, 62  
intrinsic functions, 93, 113  
    FORTRAN 77, 113  
    Fortran 95 non-standard, 108  
    Fortran 95 standard, 93

intrinsic functions (*Continued*)

    interval arithmetic, 109  
    MPI\_SIZEOF, 112  
    other vendor functions, 110  
    VMS Fortran, 132-136  
**iq\_finite(x)**, 23  
**iq\_fp\_class(x)**, 23  
**iq\_isinf(x)**, 23  
**iq\_isnan(x)**, 23  
**iq\_isnormal(x)**, 23  
**iq\_issubnormal(x)**, 23  
**iq\_iszero(x)**, 23  
**iq\_logb(x)**, 23  
**iq\_signbit(x)**, 23  
**ir\_finite(x)**, 17  
**ir\_fp\_class(x)**, 17  
**ir\_rint(x)**, 17  
**ir\_isinf(x)**, 17  
**ir\_isnan(x)**, 17  
**ir\_isnormal(x)**, 17  
**ir\_issubnormal(x)**, 17  
**ir\_iszero(x)**, 17  
**ir\_logb(x)**, 17  
**ir\_signbit(x)**, 17  
**irand**, 73  
**isatty**, 89-90  
**isetjmp**, 62-63  
ISO\_C\_BINDING module functions, 108

**J**

jump, **longjmp**, **isetjmp**, 62

**K**

**kill**, send signal, 58-59  
kind functions, Fortran 95 intrinsics, 96

**L**

left shift, **lshift**, 27  
**libm\_double**, 19-23

**libm\_quadruple**, 23  
**libm\_single**, 15-19  
**link**, 59  
 link to an existing file, **link**, 59  
**lnblk**, 56-57  
 local time zone, **ltime()**, 87  
 location of, a variable **loc**, 60  
 log gamma, 17  
 login name, get **getlog**, 48  
**long**, 61-62  
**longjmp**, 62-63  
**lshift**, 27  
**lstat**, 81-84  
**lstat64**, 84  
**ltime**, 85  
**ltime**, local time zone, 87  
**ltime64**, 88

**M**  
**malloc**, 63-66  
 mathematical functions  
   FORTRAN 77 intrinsics, 114, 122-123  
   Fortran 95 intrinsics, 94-95  
   VMS intrinsics, 133  
 matrix functions, Fortran 95 intrinsics, 98  
 maximum, positive integer, **inmax**, 57  
 memory, deallocate by **free**, 66  
 memory allocation, FORTRAN 77 intrinsics, 126-127  
 memory dump, 25  
 mode, of file, **access**, 25-26  
 MPI\_SIZEOF, 112  
**mvbits**, move bits, 67

## N

name  
   login, get, **getlog**, 48  
   terminal port, **ttynam**, 89-90  
**not**, 27  
 numeric functions, Fortran 95 intrinsics, 93-94

## O

**or**, 27  
 OS command, execute, **system**, 76, 77, 79, 84  
 overflow, 51

## P

permissions, **access** function, 25-26  
**perror**, 67  
**pid**, process id, **getpid**, 48  
 platforms, supported, 9  
 pointer, get file pointer, **getfilep**, 46  
 position file  
   **fseek**, **ftell**, 38-40  
   **fseeko64**, **ftello64**, 40-42  
 process  
   create copy with **fork** function, 38  
   id, get, **getpid**, 48  
   send signal to, **kill**, 58-59  
   wait for termination, **wait**, 90-91  
 put a character, **putc**, **fputc**, 69  
**putc**, 69

## Q

**q\_copysign(x)**, 23  
**q\_fabs(x)**, 23  
**q\_fmod(x)**, 23  
**q\_infinity()**, 23  
**q\_max\_normal()**, 24  
**q\_max\_subnormal()**, 24  
**q\_min\_normal()**, 24  
**q\_min\_subnormal()**, 24  
**q\_nextafter(x,y)**, 24  
**q\_quiet\_nan(n)**, 24  
**q\_remainder(x,y)**, 24  
**q\_scalbn(x,n)**, 24  
**q\_signaling\_nan(n)**, 24  
**qsort**, **qsort64**, 71-72  
 quadruple-precision **libm** functions, 23  
 quick sort, **qsort**, 71-72

- R**
- `r_acos(x)`, 16
  - `r_acosd(x)`, 16
  - `r_acosh(x)`, 16
  - `r_acosp(x)`, 16
  - `r_acospi(x)`, 16
  - `r_addran()`, 17
  - `r_addrans()`, 17
  - `r_asin(x)`, 16
  - `r_asind(x)`, 16
  - `r_asinh(x)`, 16
  - `r_asinp(x)`, 16
  - `r_asinpi(x)`, 16
  - `r_atan(x)`, 16
  - `r_atan2(x)`, 16
  - `r_atan2d(x)`, 16
  - `r_atan2pi(x)`, 16
  - `r_atand(x)`, 16
  - `r_atanh(x)`, 16
  - `r_atanp(x)`, 16
  - `r_atanpi(x)`, 16
  - `r_cbrt(x)`, 16
  - `r_ceil(x)`, 16
  - `r_erf(x)`, 16
  - `r_erfc(x)`, 16
  - `r_expml(x)`, 17
  - `r_floor(x)`, 17
  - `r_hypot(x)`, 17
  - `r_infinity()`, 17
  - `r_j0(x)`, 17
  - `r_j1(x)`, 17
  - `r_jn(n, x)`, 17
  - `r_lcran()`, 17
  - `r_lcrans()`, 17
  - `r_lgamma(x)`, 17
  - `r_log1p(x)`, 17
  - `r_log2(x)`, 17
  - `r_logb(x)`, 17
  - `r_max_normal()`, 18
  - `r_max_subnormal()`, 18
  - `r_min_normal()`, 18
  - `r_min_subnormal()`, 18
  - `r_nextafter(x,y)`, 18
  - `r_quiet_nan(n)`, 18
  - `r_remainder(x,y)`, 18
  - `r_rint(x)`, 18
  - `r_scalbn(x,n)`, 18
  - `r_shufrans()`, 17
  - `r_signaling_nan(n)`, 18
  - `r_significand(x)`, 18
  - `r_sin(x)`, 18
  - `r_sincos(x,s,c)`, 18
  - `r_sincosd(x,s,c)`, 18
  - `r_sincosp(x,s,c)`, 18
  - `r_sincospi(x,s,c)`, 18
  - `r_sind(x)`, 18
  - `r_sinh(x)`, 18
  - `r_sinp(x)`, 18
  - `r_sinpi(x)`, 18
  - `r_tan(x)`, 18
  - `r_tand(x)`, 18
  - `r_tanh(x)`, 18
  - `r_tanp(x)`, 18
  - `r_tanpi(x)`, 18
  - `r_y0(x)`, *bessel*, 18
  - `r_y1(x)`, *bessel*, 18
  - `r_yn(n,x)`, *bessel*, 18
  - rand**, 73
  - random
    - number, 17
    - values, **rand**, 73
  - read, character **getc**, **fgetc**, 43
  - remove a file, **unlink**, 90
  - reposition file
    - fseek**, **ftell**, 38-40
    - fseeko64**, **ftello64**, 40-42
  - right shift, **rshift**, 27
  - rindex**, 56
  - rounding direction, 51
  - rshift**, 27
- S**
- secs**, system time, 75
  - send signal to process, **kill**, 58-59
  - set\_io\_err\_handler**, 76-79
  - setbit**, 27
  - setjmp**, *See* **isetjmp**

shell prompts, 8  
**short**, 61-62  
**sigfpe**, 50  
 SIGFPE handling, 54  
**signal**, 80-81  
 signal a process, **kill**, 58-59  
 sine, 18  
 single-precision **libm** functions, 15  
**sleep**, 81  
 sort quick, **qsort**, 71-72  
 specific names, Fortran 95 intrinsics, 102-104  
**stat**, 81-84  
**stat64**, 84  
 status  
     file, **stat**, 81-84  
     file, **stat64**, 84  
     termination, **exit**, 36  
 substring, find, **index**, 56  
**SUN\_IO\_HANDLERS**, module subroutines, 76  
 supported platforms, 9  
 suspend execution for an interval, **sleep**, 81  
 symbolic, link to an existing file, **symlink**, 59  
**symlink**, 59  
**system**, 76, 77, 79, 84  
**system.inc** include file, 14  
 system time  
     **secnds**, 76  
     **time**, 85

## T

tangent, 18  
**tarray()** values for various time routines, 88  
 terminal, port name, **ttynam**, 89-90  
 terminate  
     wait for process to terminate, **wait**, 90-91  
     with status, **exit**, 36  
     write memory to core file, 25  
 time, 34-36  
     **secnds**, 76  
**time**, standard version, 86  
**time**, get system time, 85  
 trapping, floating-point, 51

trigonometric functions  
     FORTRAN 77 intrinsics, 120-122  
     VMS intrinsics, 133-134  
**ttynam**, 89-90  
 typographic conventions, 7-8

## U

underflow, 51  
**unlink**, 90  
 user ID, get, **getuid**, 48-49

## V

vector functions, Fortran 95 intrinsics, 98  
 VMS Fortran, intrinsic functions, 132-136

## W

**wait**, 90-91  
 write a character **putc**, **fputc**, 69

## X

**xknown\_lib=blas**, 108  
**xor**, 27

