Sun Studio 12 : Fortran 库参考

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

文件号码 820–1202–10

070722@18135

目录

前言

《Fortran 库参考》介绍 Sun™ Studio Fortran 库中的内函数和例程。本参考手册适用于精通 Fortran 语言和 Solaris™ 操作环境的程序员。

本指南适用于精通 Fortran 语言并希望了解如何有效使用 Sun Fortran 编译器的科学家、工程师和程序员。通常，还假定他们熟悉 Solaris 操作环境或 UNIX®。

随附的《Fortran 编程指南》中提供了有关在 Solaris 操作环境上进行 Fortran 编程的问题的讨论（包括输入/输出、应用程序开发、库的创建和使用、程序分析、移植、优化和并行处理）。

印刷约定

表 P–1 字体约定

字体	含义	示例
`AaBbCc123`	命令、文件和目录的名称；计算机屏幕输出	编辑 `.login` 文件。 使用 `ls -a` 列出所有文件。 `% You have mail.`
`AaBbCc123`	用户键入的内容，与计算机屏幕输出的显示不同	`%` **`su`** `Password:`
AaBbCc123	要使用实名或值替换的命令行占位符文本	要删除文件，请键入 **`rm`** *filename*。
AaBbCc123	保留未译的新词或术语以及要强调的词	这些称为 *class* 选项。
新词术语强调	新词或术语以及要强调的词	您**必须**成为超级用户才能执行此操作。
《书名》	书名	阅读《用户指南》的第 6 章。

- 符号 ∇ 表示有意义的空格：

```
∇∇36.001
```

- FORTRAN 77 标准采用早期惯例，名称 "FORTRAN" 的拼写采用大写字母。当前惯例是使用小写字母："Fortran 95"
- 对联机手册页的引用采用主题名称和章节号。例如，对库例程 `GETENV` 的引用为 `getenv(3F)`，这意味着访问此手册页的 man 命令为：`man -s 3F getenv`

表 P–2 代码约定

代码符号	含义	表示法	代码示例
[ ]	方括号中的参数是可选参数。	`O[`*n*`]`	`-O4, -O`
{ }	大括号中是针对所需选项的一组选择内容。	`d{y|n}`	`-dy`
|	"|" 或 "-" 符号用于分隔多个参数，只能选择其中一个参数。	`B{dynamic|static}`	`-Bstatic`
:	冒号与逗号类似，有时用于分隔多个参数。	`R`*dir*`[:`*dir*`]`	`-R/local/libs:/U/a`
…	省略号表示一系列省略。	`-xinline=`*f1*`[,…`*fn*`]`	`-xinline=alpha,dos`

Shell 提示符

Shell	提示符
C shell	*machine-name*`%`
C shell 超级用户	*machine-name*`#`
Bourne shell、Korn shell 和 GNU Bourne-Again shell	`$`
Bourne shell、Korn shell 和 GNU Bourne-Again shell 超级用户	`#`

受支持的平台

此 Sun Studio 发行版支持使用 SPARC® 和 x86 系列处理器体系结构的系统：UltraSPARC™、SPARC64、AMD64、Pentium 和 Xeon EM64T。可从以下位置获得硬件兼容性列表，在列表中可以查看您正在使用的 Solaris 操作系统版本所支持的系统：`http://www.sun.com/bigadmin/hcl`。这些文档中给出了平台类型间所有实现的区别。

在本文档中，与 x86 相关的术语的含义如下：

- "x86" 泛指 64 位和 32 位的 x86 兼容产品系列。
- "x64" 表示有关 AMD64 或 EM64T 系统的特定 64 位信息。

- “32 位 x86”表示有关基于 x86 的系统的特定 32 位信息。

有关受支持的系统，请参阅硬件兼容性列表。

访问 Sun Studio 文档

可以访问以下位置的文档：

- 可以通过随软件一起安装在本地系统或网络上的文档索引（在 Solaris 平台上为 `file:/opt/SUNWspro/docs/index.html`；在 Linux 平台上为 `file:/opt/sun/sunstudio12/docs/index.html`）获取文档。

  如果该软件没有安装在 `/opt` 目录（Solaris 平台）中或 `/opt/sun` 目录（Linux 平台）中，请询问系统管理员以获取系统中的等效路径。
- 可以从 `docs.sun.com`[sm] Web 站点获取大多数手册。下列书目只能从 Solaris 平台上已安装的软件中获取：
  - 《标准 C++ 库类参考》
  - 《标准 C++ 库用户指南》
  - 《Tools.h++ 类库参考》
  - 《Tools.h++ 用户指南》

  Solaris 平台和 Linux 平台的相应发行说明可从 `docs.sun.com` Web 站点获取。
- IDE 所有组件的联机帮助可通过 IDE 中的“帮助”菜单以及许多窗口和对话框上的“帮助”按钮获取。

可以通过 Internet 访问 `docs.sun.com` Web 站点 (`http://docs.sun.com`) 阅读、打印和购买 Sun Microsystems 的各种手册。如果找不到某手册，请参见随软件一起安装在本地系统或网络上的文档索引。

注 – Sun 对本文档中提到的第三方 Web 站点的可用性不承担任何责任。对于此类站点或资源中的（或通过它们获得的）任何内容、广告、产品或其他资料，Sun 并不表示认可，也不承担任何责任。对于因使用或依靠此类站点或资源中的（或通过它们获得的）任何内容、产品或服务而造成的或连带产生的实际或名义损坏或损失，Sun 概不负责，也不承担任何责任。

采用易读格式的文档

该文档以易读格式提供，以方便残障用户使用辅助技术进行阅读。可以按照下表所述找到文档的易读版本。如果该软件未安装在 `/opt` 目录中，请询问系统管理员以获取系统中的等效路径。

文档类型	易读版本的格式和位置
手册（第三方手册除外）	HTML，位于 http://docs.sun.com
第三方手册： ■ 《标准 C++ 库类参考》 ■ 《标准 C++ 库用户指南》 ■ 《Tools.h++ 类库参考》 ■ 《Tools.h++ 用户指南》	HTML，位于 Solaris 平台上已安装软件中，可通过文档索引 (file:/opt/SUNWspro/docs/index.html) 获取
自述文件	HTML，位于 Sun Developer Network 门户 http://developers.sun.com/sunstudio/documentation/ss12m
手册页	HTML，位于已安装软件中，可通过文档索引（在 Solaris 平台上为 file:/opt/SUNWspro/docs/index.html；在 Linux 平台上为 file:/opt/sun/sunstudio12/docs/index.html）获取
联机帮助	HTML，可通过 IDE 中的“帮助”菜单和“帮助”按钮获取
发行说明	HTML，位于 http://docs.sun.com

相关 Sun Studio 文档

下表列出了可通过 file:/opt/SUNWspro/docs/index.html 和 http://docs.sun.com 获取的相关文档。如果该软件未安装在 /opt 目录中，请询问系统管理员以获取系统中的等效路径。

文档标题	说明
《Fortran 编程指南》	介绍了如何在 Solaris 环境中编写高效 Fortran 程序；并介绍了输入/输出、库、性能、调试和并行处理。
《Fortran 用户指南》	介绍了 f95 编译器的编译时环境和命令行选项。
《OpenMP API 用户指南》	概括介绍了 OpenMP 多重处理 API，并提供了有关实现的具体信息。
《数值计算指南》	介绍了与浮点计算的数值精度有关的问题。

访问 Solaris 相关文档

下表列出了可从 docs.sun.com Web 站点上获取的相关文档。

文档集合	文档标题	说明
Solaris Reference Manual Collection	请参见手册页各章节的标题。	提供 Solaris 操作系统的有关信息。
Solaris Software Developer Collection	《链接程序和库指南》	介绍了 Solaris 链接编辑器和运行时链接程序的操作。
Solaris Software Developer Collection	《多线程编程指南》	介绍了 POSIX 和 Solaris 线程 API、使用同步对象进行编程、编译多线程程序和多线程程序的查找工具。

开发者资源

访问 Sun Developer Network Sun Studio 门户 (`http://developers.sun.com/sunstudio`) 查看下列经常更新的资源：

- 有关编程技术和最佳做法的文章
- 软件文档以及随软件一起安装的文档的更正信息
- 有关支持级别的信息
- 用户论坛
- 可下载的代码示例
- 新技术预览

Sun Studio 门户是 Sun Developer Network Web 站点 (`http://developers.sun.com`) 上面向开发者的众多其他资源之一。

联系 Sun 技术支持

如果您遇到通过本文档无法解决的技术问题，请访问以下 URL：

`http://www.sun.com/service/contacting`

Sun 欢迎您提出意见

Sun 致力于提高其文档的质量，并十分乐意收到您的意见和建议。您可以通过以下 URL 向 Sun 提交您的意见：

`http://www.sun.com/hwdocs/feedback`

请在电子邮件的主题行中注明文档的文件号码。例如，本文档的文件号码是 820-1202-10。

◆ ◆ ◆ 第 1 章

1 Fortran 库例程

本章介绍 Fortran 库例程。

本章介绍的所有例程在手册库的第 3F 节中都有对应的手册页。例如，执行 **man -s 3F access** 将显示库例程 **access** 的手册页条目。

本章没有介绍 Fortran 95 标准内例程。有关内例程的信息，请参见有关的 Fortran 95 标准文档。

另请参见《数值计算指南》，了解有关 可从 Fortran 和 C 调用的其他数学例程，其中包括 **libm** 和 **libsunmath** 中的标准数学库例程（请参见 **Intro**(3M)）、这些库的优化版本、SPARC 向量数学库、**libmvec** 以及其他库例程。

有关 **f95** 编译器实现的 Fortran 77 和 VMS 内函数的详细信息，请参见第 109 页中的“2.3.4.2 内存函数”。

1.1 数据类型注意事项

除非另有说明，否则此处列出的函数例程均不是内函数例程。这意味着函数返回的数据类型可能与函数名称的隐式类型处理相冲突，需要用户进行显式类型声明。例如，**getpid()** 返回的是 **INTEGER*4**，这就需要有 **INTEGER*4 getpid** 声明，以确保能够正确处理结果。（如果没有显式类型处理，缺省情况下，会假设为 **REAL**，这是因为函数名称以 **g** 开头。）请注意，显式类型语句位于这些例程的函数摘要中。

请注意，**IMPLICIT** 语句以及 **-dbl** 和 **-xtypemap** 编译器选项也会更改参数的数据类型处理以及返回值的处理。如果调用这些库例程时，预期的数据类型与实际的数据类型不一致，可能会导致出现意外行为。选项 **-xtypemap** 和 **-dbl** 可将 **INTEGER** 函数、**REAL** 函数和 **DOUBLE** 函数的数据类型分别提升为 **INTEGER*8**、**REAL*8** 和 **REAL*16**。为了防止出现这些问题，库调用中出现的函数名和变量必须按照其预期大小进行显式类型处理，如下所示：

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

本示例中，进行了显式类型处理，以防止使用编译器选项 **-xtypemap** 和 **-dbl** 时，库调用中出现任何数据类型提升。如果没有进行显式类型处理，这些选项可能会导致出现意外结果。有关这些选项的详细信息，请参见《Fortran 用户指南》和 **f95**(1) 手册页。

Fortran 95 编译器 **f95** 提供了一个 include 文件 **system.inc**，该文件定义了用于大多数非内库例程的接口。将该文件包含进来是为了确保对调用的函数及其参数正确地进行类型处理，尤其是在使用 **-xtypemap** 更改缺省的数据类型时。

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

可以使用 Fortran 编译器的全局程序检查选项 **-Xlist** 获取许多与库调用中出现的类型不一致有关的问题。《Fortran 用户指南》、《Fortran 编程指南》和 **f95**(1) 手册页中介绍了 **f95** 编译器执行的全局程序检查信息。

1.2 64 位环境

对程序进行编译以在 64 位操作环境中运行（也就是说，使用 **-m64** 进行编译，并在支持 64 位的 SPARC 或 x86 平台上运行可执行文件）会更改某些函数的返回值。这些通常是与标准系统级例程交互的函数（如 **malloc**(3F)）（请参见第 62 页中的 “1.4.35 **malloc**、**malloc64**、**realloc** 和 **free**：分配/重新分配/解除分配内存”），这些函数可能会根据环境采用或返回 32 位值或 64 位值。为了能够在 32 位环境与 64 位环境之间移植代码，提供了这些例程的 64 位版本，它们始终采用和/或返回 64 位值。下表列出了适用于 64 位环境的库例程：

表 1–1 适用于 64 位环境的库例程

功能	说明
malloc64	分配内存并返回指针
fseeko64	重新确定大文件的位置
ftello64	确定大文件的位置
stat64、**fstat64**、**lstat64**	确定文件的状态

表 1–1 适用于 64 位环境的库例程 （续）

功能	说明
time64、**ctime64**、**gmtime64**、**ltime64**	分解系统时间，转换为字符
qsort64	将数组元素排序

1.3 Fortran 数学函数

下列函数和子例程属于 Fortran 数学库。它们适用于使用 **f95** 编译的所有程序。这些例程属于非内例程，它们的参数采用特定数据类型并且返回值也是相同的数据类型。要使用非内例程，必须在引用它们的例程中进行声明。

其中许多例程都是“包装器”，即 C 语言库中例程的 Fortran 接口，它们本身并不是 Fortran 标准例程，其中包括 IEEE 推荐的支持函数以及专用的随机数生成器。有关这些库的更多信息，请参见《数值计算指南》以及 **libm_single**(3F)、**libm_double**(3F) 和 **libm_quadruple**(3F) 手册页。

1.3.1 单精度函数

这些子程序是单精度数学函数和子例程。

通常，下文介绍的函数可以访问单精度数学函数，它们**没有**对应的 Fortran 标准通用内函数，其数据类型按常用数据类型处理规则确定。

不必使用 **REAL** 语句对这些函数进行显式类型处理，只要保留缺省的类型处理即可。（以 "**r**" 开头的名称表示 **REAL**，以 "**i**" 开头的名称表示 **INTEGER**。）

有关这些例程的详细信息，参见 C 数学库手册页 (3M)。例如，有关 **r_acos(x)** 的信息，请参见 **acos**(3M) 手册页。

表 1–2 单精度数学函数

函数名	返回类型	说明
r_acos(x)	REAL	反余弦
r_acosd(x)	REAL	--
r_acosh(x)	REAL	反双曲余弦
r_acosp(x)	REAL	--
r_acospi(x)	REAL	--
r_atan(x)	REAL	反正切
r_atand(x)	REAL	--
r_atanh(x)	REAL	反双曲正切
r_atanp(x)	REAL	--
r_atanpi(x)	REAL	--
r_asin(x)	REAL	反正弦
r_asind(x)	REAL	--
r_asinh(x)	REAL	反双曲正弦
r_asinp(x)	REAL	--
r_asinpi(x)	REAL	--
r_atan2((y, x)	REAL	反正切
r_atan2d(y, x)	REAL	--
r_atan2pi(y, x)	REAL	--
r_cbrt(x)	REAL	立方根
r_ceil(x)	REAL	计算大于或等于 x 的最小整数
r_copysign(x, y)	REAL	--
r_cos(x)	REAL	余弦
r_cosd(x)	REAL	--
r_cosh(x)	REAL	双曲余弦
r_cosp(x)	REAL	--
r_cospi(x)	REAL	--
r_erf(x)	REAL	误差函数
r_erfc(x)	REAL	--

表 1–2 单精度数学函数 （续）

函数名	返回类型	说明
r_expm1(x)	REAL	(e**x)-1
r_floor(x)	REAL	计算不大于 x 的下一个整数
r_hypot(x, y)	REAL	计算直角三角形的斜边长度
r_infinity( )	REAL	--
r_j0(x)	REAL	贝塞尔--
r_j1(x)	REAL	--
r_jn(n, x)	REAL	
ir_finite(x)	INTEGER	--
ir_fp_class(x)	INTEGER	--
ir_ilogb(x)	INTEGER	--
ir_irint(x)	INTEGER	--
ir_isinf(x)	INTEGER	--
ir_isnan(x)	INTEGER	--
ir_isnormal(x)	INTEGER	--
ir_issubnormal(x)	INTEGER	--
ir_iszero(x)	INTEGER	--
ir_signbit(x)	INTEGER	--
r_addran()	REAL	随机数生成器
r_addrans(x, p, l, u)	子例程 REAL	
r_lcran()	子例程	
r_lcrans(x, p, l, u)	子例程	
r_shufrans(x, p, l, u)		
r_lgamma(x)	REAL	gamma(x) 的对数
r_logb(x)	REAL	--
r_log1p(x)	REAL	--
r_log2(x)	REAL	--

表 1–2 单精度数学函数 （续）

函数名	返回类型	说明
r_max_normal()	REAL	
r_max_subnormal()	REAL	
r_min_normal()	REAL	
r_min_subnormal()	REAL	
r_nextafter(x, y)	REAL	
r_quiet_nan(n)	REAL	
r_remainder(x, y)	REAL	
r_rint(x)	REAL	
r_scalb(x, y)	REAL	
r_scalbn(x, n)	REAL	
r_signaling_nan(n)	REAL	
r_significand(x)	REAL	
r_sin(x)	REAL	正弦
r_sind(x)	REAL	--
r_sinh(x)	REAL	双曲正弦
r_sinp(x)	REAL	--
r_sinpi(x)	REAL	--
r_sincos(x, s, c)	子例程	正弦和余弦
r_sincosd(x, s, c)	子例程	--
r_sincosp(x, s, c)	子例程	--
r_sincospi(x, s, c)	子例程	--
r_tan(x)	REAL	正切
r_tand(x)	REAL	--
r_tanh(x)	REAL	双曲正切
r_tanp(x)	REAL	--
r_tanpi(x)	REAL	--
r_y0(x)	REAL	贝塞尔
r_y1(x)	REAL	--
r_yn(n, x)	REAL	--

- 变量 c、l、p、s、u、x 和 y 的类型为 REAL。变量 n 的类型为 INTEGER。

- 如果使用了会将名称以 "**r**" 开头的函数声明为另外一种数据类型的 **IMPLICIT** 语句，则应将这些函数的类型显式声明为 **REAL**。
- **sind(x)** 和 **asind(x)** 等函数采用**度数**，而不是**弧度**。

另请参见：**intro**(3M) 和《数值计算指南》。

1.3.2 双精度函数

以下子程序为双精度数学函数和子例程。

通常，这些函数**没有**对应的 Fortran 标准通用内函数，其数据类型按常用数据类型处理规则确定。

这些 **DOUBLE PRECISION** 函数应该用于 **DOUBLE PRECISION** 语句中。

有关详细信息，请参阅 C 库手册页：**acos**(3M) 手册页中介绍了 **d_acos(x)**。

表 1–3 双精度数学函数

函数名	返回类型	说明
d_acos(x)	**DOUBLE PRECISION**	反余弦
d_acosd(x)	**DOUBLE PRECISION**	--
d_acosh(x)	**DOUBLE PRECISION**	反双曲余弦
d_acosp(x)	**DOUBLE PRECISION**	--
d_acospi(x)	**DOUBLE PRECISION**	--
d_atan(x)	**DOUBLE PRECISION**	反正切
d_atand(x)	**DOUBLE PRECISION**	--
d_atanh(x)	**DOUBLE PRECISION**	反双曲正切
d_atanp(x)	**DOUBLE PRECISION**	--
d_atanpi(x)	**DOUBLE PRECISION**	--
d_asin(x)	**DOUBLE PRECISION**	反正弦
d_asind(x)	**DOUBLE PRECISION**	--
d_asinh(x)	**DOUBLE PRECISION**	反双曲正弦
d_asinp(x)	**DOUBLE PRECISION**	--
d_asinpi(x)	**DOUBLE PRECISION**	--

表 1–3 双精度数学函数 （续）

函数名	返回类型	说明
d_atan2((y, x)	DOUBLE PRECISION	反正切
d_atan2d(y, x)	DOUBLE PRECISION	--
d_atan2pi(y, x)	DOUBLE PRECISION	--
d_cbrt(x)	DOUBLE PRECISION	立方根
d_ceil(x)	DOUBLE PRECISION	计算大于或等于 x 的最小整数
d_copysign(x, x)	DOUBLE PRECISION	--
d_cos(x)	DOUBLE PRECISION	余弦
d_cosd(x)	DOUBLE PRECISION	--
d_cosh(x)	DOUBLE PRECISION	双曲余弦
d_cosp(x)	DOUBLE PRECISION	--
d_cospi(x)	DOUBLE PRECISION	--
d_erf(x)	DOUBLE PRECISION	误差函数
d_erfc(x)	DOUBLE PRECISION	--
d_expm1(x)	DOUBLE PRECISION	(e**x)-1
d_floor(x)	DOUBLE PRECISION	计算不大于 x 的下一个整数
d_hypot(x, y)	DOUBLE PRECISION	计算直角三角形的斜边长度
d_infinity( )	DOUBLE PRECISION	--
d_j0(x)	DOUBLE PRECISION	贝塞尔
d_j1(x)	DOUBLE PRECISION	--
d_jn(n, x)	DOUBLE PRECISION	--

表 1–3 双精度数学函数 （续）

函数名	返回类型	说明
id_finite(x)	INTEGER	
id_fp_class(x)	INTEGER	
id_ilogb(x)	INTEGER	
id_irint(x)	INTEGER	
id_isinf(x)	INTEGER	
id_isnan(x)	INTEGER	
id_isnormal(x)	INTEGER	
id_issubnormal(x)	INTEGER	
id_iszero(x)	INTEGER	
id_signbit(x)	INTEGER	
d_addran()	DOUBLE PRECISION	随机数生成器
d_addrans(x, p, l, u)	子例程	
d_lcran()	DOUBLE PRECISION	
d_lcrans(x, p, l, u)	子例程	
d_shufrans(x, p, l,u)	子例程	
d_lgamma(x)	DOUBLE PRECISION	gamma(x) 的对数
d_logb(x)	DOUBLE PRECISION	--
d_log1p(x)	DOUBLE PRECISION	--
d_log2(x)	DOUBLE PRECISION	--

表 1–3 双精度数学函数 （续）

函数名	返回类型	说明
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	正弦
d_sind(x)	DOUBLE PRECISION	--
d_sinh(x)	DOUBLE PRECISION	双曲正弦
d_sinp(x)	DOUBLE PRECISION	--
d_sinpi(x)	DOUBLE PRECISION	--
d_sincos(x, s, c)	子例程	正弦和余弦
d_sincosd(x, s, c)	子例程	--
d_sincosp(x, s, c)	子例程	--
d_sincospi(x, s, c)	子例程	
d_tan(x)	DOUBLE PRECISION	正切
d_tand(x)	DOUBLE PRECISION	--
d_tanh(x)	DOUBLE PRECISION	双曲正切
d_tanp(x)	DOUBLE PRECISION	--
d_tanpi(x)	DOUBLE PRECISION	--
d_y0(x)	DOUBLE PRECISION	贝塞尔
d_y1(x)	DOUBLE PRECISION	--
d_yn(n, x)	DOUBLE PRECISION	--

- 变量 c、l、p、s、u、x 和 y 的类型为 DOUBLE PRECISION。变量 n 的类型为 INTEGER。

- 应在 **DOUBLE PRECISION** 语句中或者通过适当的 **IMPLICIT** 语句显式声明这些函数的类型。
- **sind(x)** 和 **asind(x)** 等函数采用**度数**，而不是**弧度**。

另请参见：**intro**(3M) 和《数值计算指南》。

1.3.3 四倍精度函数

以下子程序为**四倍**精度 (**REAL*16**) 数学函数和子例程。

通常，这些函数**没有**对应的标准通用内函数，其数据类型按常用数据类型处理规则确定。

四倍精度函数必须用于 **REAL*16** 语句中。

表 1–4 四倍精度 **libm** 函数

函数名	返回类型
q_copysign(x, y)	**REAL*16**
q_fabs(x)	**REAL*16**
q_fmod(x)	**REAL*16**
q_infinity( )	**REAL*16**
iq_finite(x)	**INTEGER**
iq_fp_class(x)	**INTEGER**
iq_ilogb(x)	**INTEGER**
iq_isinf(x)	**INTEGER**
iq_isnan(x)	**INTEGER**
iq_isnormal(x)	**INTEGER**
iq_issubnormal(x)	**INTEGER**
iq_iszero(x)	**INTEGER**
iq_signbit(x)	**INTEGER**

表 1–4 四倍精度 **libm** 函数 （续）

函数名	返回类型
q_max_normal()	**REAL*16**
q_max_subnormal()	**REAL*16**
q_min_normal()	**REAL*16**
q_min_subnormal()	**REAL*16**
q_nextafter(x, y)	**REAL*16**
q_quiet_nan(n)	**REAL*16**
q_remainder(x, y)	**REAL*16**
q_scalbn(x, n)	**REAL*16**
q_signaling_nan(n)	**REAL*16**

- 变量 **c**、**l**、**p**、**s**、**u**、**x** 和 **y** 的类型为四倍精度。变量 **n** 的类型为 **INTEGER**。
- 应使用 **REAL*16** 语句或者通过适当的 **IMPLICIT** 语句显式声明这些函数的类型。
- **sind(x)** 和 **asind(x)** 等函数采用**度数**，而不是**弧度**。

如果需要使用其他任何四倍精度 **libm** 函数，可以在调用前使用 **$PRAGMA C**(*fcn*) 来进行调用。有关详细信息，请参见《Fortran 编程指南》中介绍 C–Fortran 接口的章节。

1.4 Fortran 库例程参考

本节详细介绍了 Fortran 库中属于 Sun Studio Fortran 95 软件但不是 Fortran 95 标准内例程和函数的子例程和函数。

下面以表的形式概要说明调用接口

数据声明			
带参数的调用原型概要说明			
参数 *1* **名称**	**数据类型**	**输入/输出**	**说明**
参数 *2* **名称**	**数据类型**	**输入/输出**	**说明**
返回值	**数据类型**	输出	**说明**

可以在 Sun Studio 手册页的第 3f 节中找到其他手册页。例如，执行 **man -s 3f** 命令将显示 **access()** 函数的手册页。在本手册中，手册页参考均表示为 *manpagename*(*section*)。例如，有关 **access()** 函数的手册页参考表示为 **access**(3f)，有关 Fortran 95 编译器的手册页参考表示为 **f95**(1)。

1.4.1 abort：终止并写入核心转储文件

该子例程的调用方式如下所示：

```
call abort
```

abort 刷新 I/O 缓冲区，然后中止进程，可能会在当前目录中生成**核心转储**文件内存转储。有关限制或抑制核心转储的信息，请参见 **limit**(1)。

1.4.2 access：检查文件权限或文件是否存在

该函数的调用方式如下所示：

INTEGER*4 access

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

name	**字符**	输入	文件名
mode	**字符**	输入	权限
返回值	**INTEGER*4**	输出	*status*=0：OK；*status*>0：错误代码

access 确定是否可以使用 *mode* 指定的权限访问文件 *name*。如果可以使用 *mode* 指定的权限成功访问文件，access 将返回零。另请参见 **gerror**(3F)，了解有关错误代码的信息。

可以将 *mode* 设置为 **r**、**w** 和 **x** 中的一个或多个（以任何顺序或任意组合），也可以为空白，其中 **r**、**w** 和 **x** 的含义如下：

'r'	测试是否有读取权限
'w'	测试是否有写入权限
'x'	测试是否有执行权限
' '	测试文件是否存在

示例 1：测试是否有读/写权限：

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

示例 2：测试文件是否存在：

```
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：在指定的时间后调用子例程

该函数的调用方式如下所示：

INTEGER*4 alarm

n = alarm (*time*, *sbrtn* **)**

time	**INTEGER*4**	输入	等待的秒数（0=不调用）
sbrtn	例程名称	输入	要执行的子程序必须列在外部语句中
返回值	**INTEGER*4**	输出	最后一次报警的剩余时间

示例：**alarm**－等待 9 秒后调用 **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
```

另请参见：**alarm**(3C)、**sleep**(3F) 和 **signal**(3F)。注意以下限制条件：

- 子例程无法将自己的名称传递给 **alarm**。
- **alarm** 例程生成的信号可能会妨碍 I/O 操作。因此被调用的子例程 *sbrtn* 本身不得执行任何 I/O 操作。
- 从并行或多线程 Fortran 程序中调用 **alarm()** 可能会产生不可预料的结果。

1.4.4 bit：位函数 and、or、...、bit、setbit、...

定义如下：

and(*word1*, *word2* **)**	对其参数进行按位**与**运算。
or(*word1*, *word2* **)**	对其参数进行按位**或**运算。
xor(*word1*, *word2* **)**	对其参数进行按位**异或**运算。
not(*word* **)**	返回其参数的按位**补**结果。
lshift(*word*, *nbits* **)**	结果不循环进位的逻辑左移。
rshift(*word*, *nbits* **)**	带符号扩展的算术右移。
call bis(*bitnum*, *word* **)**	将 *word* 中的 *bitnum* 位设置为 1。
call bic(*bitnum*, *word* **)**	将 *word* 中的 *bitnum* 位清除为 0。
bit(*bitnum*, *word* **)**	测试 *word* 中的*bitnum* 位，如果该位是 1，则返回 **LOGICAL .true.**，如果该位是 0，则返回 **.false.**。
call setbit(*bitnum*,*word*,*state***)**	如果 *state* 是非零值，则将 *word* 中的 *bitnum* 位设置为 1，否则将其清除。

MIL-STD-1753 的另一可换用外部版本为：

iand(*m*, *n* **)**	对其参数进行按位**与**运算。
ior(*m*, *n* **)**	对其参数进行按位**或**运算。
ieor(*m*, *n* **)**	对其参数进行按位**异或**运算。
ishft(*m*, *k* **)**	结尾不循环进位的逻辑移位（如果 *k*>0 则为左移，如果 *k*<0 则为右移）。
ishftc(*m*, *k*, *ic* **)**	循环移位：*m* 最右边的 *ic* 位循环左移 *k* 个位置。
ibits(*m*, *i*, *len* **)**	提取位：从 *m* 中的 *i* 位开始提取 *len* 个位。
ibset(*m*, *i* **)**	设置位：返回值等于字 *m*，且位数 *i* 设置为 1。
ibclr(*m*, *i* **)**	清除位：返回值等于字 *m*，且位数 *i* 设置为 0。
btest(*m*, *i* **)**	测试 *m* 中的 *i* 位；如果该位是 1，则返回 **LOGICAL .true.**，如果该位是 0，则返回 **.false.**。

有关对位字段进行操作的其他函数，另请参见第 65 页中的 “1.4.36 **mvbits**：移动位字段” 以及第 2 章和第 3 章。

1.4.4.1 用法：and、or、xor、not、rshift 和 lshift

对于内函数：

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* 和 *nbits* 都是整型输入参数。这些函数是编译器内联扩展的内函数。返回值的数据类型是第一个参数的数据类型。

不测试 *nbits* 的值是否合理。

示例：**and**、**or**、**xor** 和 **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)
 00000000004 00000000007 00000000003 37777777773
demo%
```

示例：**lshift** 和 **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 用法：bic、bis、bit 和 setbit

对于子例程和函数

call bic(*bitnum*, *word* **)**
call bis(*bitnum*, *word* **)**
call setbit(*bitnum*, *word*, *state* **)**
LOGICAL bit
x **= bit(** *bitnum*, *word* **)**

bitnum、*state* 和 *word* 都是 **INTEGER*4** 输入参数。函数 **bit**() 的返回值是逻辑值。

各个位都进行编号，0 位表示最低有效位，31 位表示最高有效位。

bic、**bis** 和 **setbit** 是外部子例程，**bit** 是外部函数。

示例 3：**bic**、**bis**、**setbit** 和 **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：更改缺省目录

该函数的调用方式如下所示：

INTEGER*4 chdir

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

dirname	**字符**	输入	目录名称
返回值	**INTEGER*4**	输出	*n*=0：OK；*n*>0：错误代码

示例：**chdir**－将**cwd** 更改为 **MyDir**：

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

另请参见：**chdir**(2)、**cd**(1) 和 **gerror**(3F)，了解有关错误代码的信息。

路径名长度不能超过 **<sys/param.h>** 中定义的 **MAXPATHLEN** 值。路径可以是相对路径，也可以是绝对路径。

使用该函数可能会导致按单元查询失败。

某些 Fortran 文件操作会按文件名重新打开文件。执行 I/O 操作时使用 **chdir** 可能会导致运行时系统不能跟踪使用相对路径名创建的文件。（包括使用打开语句创建但没有文件名的文件）。

1.4.6 chmod：更改文件的模式

该函数的调用方式如下所示：

INTEGER*4 chmod

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

name	**字符**	输入	路径名
mode	**字符**	输入	*chmod*(1) 可以识别的任意字符， 例如 **o-w**、**444** 等。
返回值	**INTEGER*4**	输出	*n* = 0：OK；*n*>0：系统错误编号

示例：**chmod**－添加对 **MyFile** 的写入权限：

```
character*18 name, mode
INTEGER*4 chmod, n
name = 'MyFile'
```

```
        mode = '+w'
        n =  chmod( name, mode )
        if ( n .ne. 0 ) stop 'chmod: error'
        end
```

另请参见：**chmod**(1) 和 **gerror**(3F)，了解有关错误代码的信息。

路径名长度不能超过 **<sys/param.h>** 中定义的 **MAXPATHLEN** 值。路径可以是相对路径，也可以是绝对路径。

1.4.7 date：获取以字符串表示的当前日期

注 – 由于该例程只返回两位数值的年份，因此它存在“2000 年安全”问题。在 1999 年 12 月 31 日之后，使用该例程输出计算日期差异的程序可能无法正常工作。如果执行使用此 **date()** 例程的程序，第一次调用该例程时会显示运行时警告消息，向用户发出报警。请参见另一个可换用的例程 **date_and_time()**。

该子例程的调用方式如下所示：

call date(*c* **)**			
c	**CHARACTER*9**	输出	变量、数组、数组元素或字符子串

返回的字符串 *c* 的格式为 *dd-mmm-yy*，其中 *dd* 表示两位数的当月日期，*mmm* 表示三个字母的月份缩写，*yy* 表示两位数的年份（因此存在 2000 年安全问题！）。

示例：**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%
```

另请参见 **idate()** 和 **date_and_time()**。

1.4.7.1 date_and_time：获取日期和时间

这是一个 2000 年安全的 Fortran 95 内例程。

子例程 **date_and_time** 返回实时时钟和日期的相关数据。返回数据包括本地时间以及本地时间与通用协调时间 (Universal Coordinated Time, UTC) 之间的时差，通用协调时间也称为格林威治标准时间 (Greenwich Mean Time, GMT)。

子例程 **date_and_time()** 的调用方式如下：

call date_and_time(*date* , *time*, *zone*, *values* **)**			
date	**CHARACTER*8**	输出	以 CCYYMMDD 格式表示的日期，其中 CCYY 表示四位数的年份，MM 表示两位数的月份，DD 表示两位数的当月日期。例如：19980709
time	**CHARACTER*10**	输出	以 hhmmss.sss 格式表示的当前时间，其中 hh 表示小时，mm 表示分钟，ss.sss 表示秒和毫秒。
zone	**CHARACTER*5**	输出	与 UTC 的时差，以小时数和分钟数表示，采用 hhmm 格式。
values	**INTEGER*4 VALUES(8)**	输出	下面介绍的 8 个元素组成的整数数组。

INTEGER*4 *values* 数组中返回的 8 个值为

VALUES(1)	以 4 位整数表示的年份。例如：1998。
VALUES(2)	以从 1 到 12 的整数表示的月份。
VALUES(3)	以从 1 到 31 的整数表示的当月日期。
VALUES(4)	以分钟数表示的与 UTC 的时差。
VALUES(5)	以从 1 到 23 的整数表示的当天小时时间。
VALUES(6)	以从 1 到 59 的整数表示的一个小时中的分钟时间。
VALUES(7)	以从 0 到 60 的整数表示的一分钟中的秒数。
VALUES(8)	位于范围 0 至 999 中的毫秒数。

date_and_time 使用示例：

```
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
```

2000 年 2 月 16 日在美国加利福尼亚的一台计算机上运行该例程时，输出结果如下所示：

```
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 dtime 和 etime：已用的执行时间

这两个函数的返回值都是已用时间（如果为 -1.0，表示出现错误）。返回的时间以秒数表示。

缺省情况下，Fortran 95 使用的 **dtime** 和 **etime** 版本使用系统的低精度时钟。该精度是百分之一秒。但是，如果在 Sun OS™ 操作系统实用程序 **ptime**(1) (**/usr/proc/bin/ptime**) 下运行程序，则使用高精度时钟。

1.4.8.1 dtime：自上次调用 dtime 后已用时间

对于 **dtime**，已用时间为：

- 第一次调用：自开始执行后已用时间
- 后来调用：自上次调用 **dtime** 后已用时间
- 单处理器：CPU 占用的时间

- 多处理器：所有 CPU 占用时间总和，该数据没有什么用，这时可改用 **etime**。

注 – 在并行循环中调用 **dtime** 会得到不确定的结果，这是因为参与循环的所有线程都共用已用时间计数器。

该函数的调用方式如下所示：

e = **dtime(** *tarray* **)**			
tarray	**real(2)**	输出	*e*= -1.0：错误：*tarray* 值未定义 *e*≠ -1.0：没有错误，*tarray(1)* 中存储的是用户时间，*tarray(2)* 中存储的是系统时间
返回值	**real**	输出	*e*= -1.0：错误 *e*≠ -1.0：*tarray(1)* 与 *tarray(2)* 之和

示例：**dtime()**，单处理器：

```
demo% cat tdtime.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 tdtime.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：自开始执行后已用时间

对于 **etime**，已用时间为：

- 单处理器执行－调用进程的 CPU 时间
- 多处理器执行－处理程序时的挂钟时间

如果环境变量 **PARALLEL** 或 **OMP_NUM_THREADS** 定义为大于 1 的某个整数值，运行时库就确定程序是在多处理器模式下执行。

该函数的调用方式如下所示：

e = **etime(** *tarray* **)**			
tarray	**real(2)**	输出	*e*= -1.0：错误：*tarray* 值未定义 *e*≠ -1.0：单处理器：*tarray(1)* 中存储的是用户时间，*tarray(2)* 中存储的是系统时间 多处理器：*tarray(1)* 中存储的是挂钟时间，*tarray(2)* 中存储的是 0.0
返回值	**real**	输出	*e*= -1.0：错误 *e*≠ -1.0：*tarray(1)* 与 *tarray(2)* 之和

请注意，初次调用 **etime** 时所得结果不准确。它只是使系统时钟开始运转。请勿使用初次调用 etime 返回的值。

示例：**etime()**，单处理器：

```
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%
```

另请参见 **times**(2) 和《Fortran 编程指南》。

1.4.9 exit：终止进程并设置状态

该子例程的调用方式如下所示：

call exit(*status* **)**		
status	**INTEGER*4**	输入

示例：**exit()**：

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

exit 刷新并关闭进程中的所有文件，然后通知父进程（如果它在执行 **wait**）。

status 的低 8 位可用于父进程。此时，这 8 位左移 8 位，其他所有位均为零。（因此，*status* 应介于 256 到 65280 之间。）该调用从不返回任何值。

执行 C 函数 **exit** 时，可能会在系统最终 '**exit**' 之前执行清除操作。

调用 **exit** 时不使用参数会导致出现编译时警告消息，并自动将零作为参数。另请参见：**exit**(2)、**fork**(2)、**fork**(3F)、**wait**(2) 和 **wait**(3F)。

1.4.10 fdate：返回以 ASCII 字符串表示的日期和时间

该子例程或函数的调用方式如下所示：

call fdate(*string* **)**		
string	**character*24**	输出

或者：

CHARACTER fdate*24 *string* **= fdate()**			如果用作函数，调用例程必须定义 **fdate** 的类型和大小。
返回值	**character*24**	输出	

示例 1：**fdate** 用作子例程：

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

输出：

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

示例 2：**fdate** 用作函数，输出相同：

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

另请参见：**ctime**(3)、**time**(3F) 和 **idate**(3F)。

1.4.11 flush：刷新逻辑单元的输出

该函数的调用方式如下所示：

INTEGER*4 flush *n* = **flush(** *lunit* **)**			
lunit	**INTEGER*4**	输入	逻辑单元
返回值	**INTEGER*4**	输出	*n* = 0 表示没有错误，*n* > 0 表示错误编号

flush 函数将逻辑单元 **lunit** 的缓冲区的内容刷新到关联文件中。逻辑单元 0 和逻辑单元 6 都与控制台终端关联时，该函数对它们特别有用。如果出现错误，该函数返回正的错误编号，否则返回零。

另请参见 **fclose**(3S)。

1.4.12 fork：创建当前进程的副本

该函数的调用方式如下所示：

INTEGER*4 fork *n* = **fork()**			
返回值	**INTEGER*4**	输出	*n*>0：*n* 为副本的进程 ID *n*<0, *n* 为系统错误代码

fork 函数创建调用进程的副本。两个进程之间的唯一区别在于返回给其中一个进程（称为**父**进程）的值是副本的进程 ID。副本通常称为**子**进程。返回给子进程的值将为零。

为了避免外部文件中的 I/O 缓冲区内容重复，在执行 fork 操作之前，会刷新所有已开放供写入的逻辑单元。

示例：**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
```

目前尚未提供对应的 **exec** 例程，这是因为没有一种令人满意的方法能够在整个 **exec** 例程中保持开放的逻辑单元。但是，可以使用 **system**(3F) 执行 **fork/exec** 的常用函数。另请参见： **fork**(2)、**wait**(3F)、**kill**(3F)、**system**(3F) 和 **perror**(3F)。

1.4.13 fseek 和 ftell：确定文件的位置和复位文件

fseek 和 **ftell** 是用于实现文件复位的例程。**ftell** 返回文件的当前位置，以距文件开头的字节偏移量表示。在程序中后面某处，**fseek** 可以使用此保存的偏移值，将文件复位到原来位置以便读取。

1.4.13.1 fseek：将文件复位到逻辑单元中

该函数的调用方式如下所示：

INTEGER*4 fseek

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

lunit	**INTEGER*4**	输入	开放的逻辑单元
offset	**INTEGER*4** **或** **INTEGER*8**	输入	相对于 *from* 指定位置的字节偏移量
	如果针对 64 位环境进行了编译，需要 **INTEGER*8** 偏移值。如果提供了文字常量，它必须是 64 位常量，例如：**100_8**		
from	**INTEGER*4**	输入	0=文件开头 1=当前位置 2=文件结尾
返回值	**INTEGER*4**	输出	*n*=0：OK；*n*>0：系统错误代码

注 – 对于后续文件，在调用 **fseek** 后面执行输出操作（例如 **WRITE**）会导致 **fseek** 位置后面的所有数据记录被删除，并替换为新的数据记录（以及文件结束标记）。只有在使用直接访问文件时，才能将记录重新写入到位。

示例：**fseek()** — 将 **MyFile** 复位到距开头两个字节处

```
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
```

示例：同上，但在 64 位环境中，使用 **-m64** 编译：

```
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：返回文件的当前位置

该函数的调用方式如下所示：

INTEGER*4 ftell

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

lunit	**INTEGER*4**	输入	开放的逻辑单元
返回值	**INTEGER*4**	输出	*n*>=0：*n* 为距文件开头的字节偏移量 *n*<0：*n* 为系统错误代码

示例：**ftell()**：

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

示例：同上，但在 64 位环境中，使用 **-m64** 编译：

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

另请参见 **fseek**(3S) 和 **perror**(3F) 以及 **fseeko64**(3F) 和 **ftello64**(3F)。

1.4.14 fseeko64 和 ftello64：确定大型文件的位置和复位大型文件

fseeko64 和 **ftello64** 是 **fseek** 和 **ftell** 的“大型文件”版本。它们采用并返回 **INTEGER*8** 的文件位置偏移值。（“大型文件”是指大于 2 GB 的文件，因此字节位置必须以 64 位整型值表示。）可使用这些函数确定大型文件的位置和/或复位大型文件。

1.4.14.1 fseeko64：将文件复位到逻辑单元中

该函数的调用方式如下所示：

INTEGER fseeko64

n **= fseeko64(** *lunit*, *offset64*, *from* **)**

lunit	**INTEGER*4**	输入	开放的逻辑单元
offset64	**INTEGER*8**	输入	相对于 *from* 指定位置的 64 位字节偏移量
from	**INTEGER*4**	输入	0=文件开头 1=当前位置 2=文件结尾
返回值	**INTEGER*4**	输出	*n*=0：OK；*n*>0：系统错误代码

注 – 对于后续文件，在调用 **fseeko64** 后执行输出操作（例如 **WRITE**）会导致 **fseeko64** 位置后面的所有数据记录被删除，并替换为新的数据记录（以及文件结束标记）。只有在使用直接访问文件时，才能将记录重新写入到位。

示例：**fseeko64()** – 将 **MyFile** 复位到距开头两个字节处：

```
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：返回文件的当前位置

该函数的调用方式如下所示：

INTEGER*8 ftello64			
n = **ftell**o64**(** *lunit* **)**			
lunit	**INTEGER*4**	输入	开放的逻辑单元
返回值	**INTEGER*8**	输出	*n*≥0：*n* 为距文件开头的字节偏移量 *n*<0：*n* 为系统错误代码

示例：**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 和 iargc：获取命令行参数

getarg 和 **iargc** 访问命令行上的参数（在命令行预处理程序扩展后）。

1.4.15.1 getarg：获取命令行参数

该子例程的调用方式如下所示：

call getarg(*k*, *arg* **)**			
k	**INTEGER*4**	输入	参数索引（0=第一个=命令名称）
arg	**character***n*	输出	第 *k* 个参数
n	**INTEGER*4**	*arg* 的大小	大得足以容纳最长的参数

1.4.15.2 iargc：获取命令行参数的数量

该函数的调用方式如下所示：

m = **iargc()**			
返回值	**INTEGER*4**	输出	命令行中参数的数量

示例：使用 **iargc** 和 **getarg** 获取参数的数量和每个参数：

```
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
```

另请参见 **execve**(2) 和 **getenv**(3F)。

1.4.16 getc 和 fgetc：获取下一个字符

getc 和 **fgetc** 从输入流中获取下一个字符。请勿将这些例程的调用与相同逻辑单元中进行的正常 Fortran I/O 混在一起。

1.4.16.1 getc：从 stdin 中获取下一个字符

该函数的调用方式如下所示：

INTEGER*4 getc *status* = **getc(** *char* **)**			
char	**字符**	输出	下一个字符
返回值	**INTEGER*4**	输出	*status*=0：OK *status*=-1：文件结束 *status*>0：系统错误代码或 **f95** I/O 错误代码

示例：使用 **getc** 获取从键盘输入的每个字符；请注意 Ctrl-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
```

编译之后，运行以上源代码的样例如下：

```
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%
```

对于逻辑单元，请勿将正常的 Fortran 输入与 **getc()** 混在一起。

1.4.16.2 fgetc：从指定逻辑单元中获取下一个字符

该函数的调用方式如下所示：

INTEGER*4 fgetc

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

lunit	**INTEGER*4**	输入	逻辑单元
char	**字符**	输出	下一个字符
返回值	**INTEGER*4**	输出	*status*=-1：文件结束 *status*>0：系统错误代码或 **f95** I/O 错误代码

示例：使用 **fgetc** 从 **tfgetc.data** 中获取每个字符；请注意换行 (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
```

编译之后，运行以上源代码的样例如下：

```
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%
```

对于逻辑单元，请勿将正常的 Fortran 输入与 **fgetc()** 混在一起。

另请参见：**getc**(3S)、**intro**(2) 和 **perror**(3F)。

1.4.17 getcwd：获取当前工作目录的路径

该函数的调用方式如下所示：

INTEGER*4 getcwd

status **= getcwd(** *dirname* **)**

dirname	**character***n	输出 返回当前目录的路径	当前工作目录的路径名称。*n* 必须足够大，以便能容纳最长的路径名称
返回值	**INTEGER*4**	输出	*status*=0：OK *status*>0：错误代码

示例：**getcwd**：

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

另请参见：**chdir**(3F)、**perror**(3F) 和 **getwd**(3)。

注意：路径名长度不能超过 **<sys/param.h>** 中定义的 **MAXPATHLEN** 值。

1.4.18 getenv：获取环境变量值

该子例程的调用方式如下所示：

call getenv(*ename*, *evalue* **)**			
ename	**character*** *n*	输入	寻找的环境变量名称
evalue	**character*** *n*	输出	找到的环境变量值，如果不成功，则为空白。

ename 和 *evalue* 的大小必须足够大，以便能容纳相应的字符串。

如果 *evalue* 太短而不能容纳整个字符串值，则字符串将被截断以符合 *evalue*。

子例程 **getenv** 在环境列表中搜索格式为 *ename=evalue* 的字符串，如果找到，则通过 *evalue* 返回值；否则在 *evalue* 中填入空白。

示例：使用 **getenv()** 打印 **$SHELL** 值：

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

另请参见：*execve*(2) 和 *environ*(5)。

1.4.19 getfd：获取外部单元编号的文件描述符

该函数的调用方式如下所示：

INTEGER*4 getfd *fildes* **= getfd(** *unitn* **)**			
unitn	**INTEGER*4**	输入	外部单元编号
返回值	**INTEGER*4** **或 INTEGER*8**	输出	如果已连接文件，则返回文件描述符；如果未连接文件，则返回 -1。如果针对 64 位环境进行了编译，返回 **INTEGER*8** 值

示例：**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
```

另请参见 **open(2)**。

1.4.20 getfilep：获取外部单元编号的文件指针

该函数为：

irtn = *c_read*(**getfilep**(*unitn*), *inbyte*, **1**)			
c_read	C 函数	输入	用户自己的 C 函数。请参见示例。
unitn	**INTEGER*4**	输入	外部单元编号。
getfilep	**INTEGER*4** **或 INTEGER*8**	返回值	如果已连接文件，则返回文件指针；如果未连接文件，则返回 -1。如果针对 64 位环境进行了编译，则返回 **INTEGER*8** 值

该函数用于将标准的 Fortran I/O 与 C I/O 混在一起。此类混合不可移植，也不保证在使用后续发行版本的操作系统或 Fortran 时可以进行此操作。建议不要使用该函数，并且没有提供直接的接口。必须创建自己的 C 例程，才能使用 **getfilep** 返回的值。下面所示是一个 C 例程样例。

示例：Fortran 通过将 **getfilep** 传递给 C 函数来使用 getfile：

```
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
```

实际使用 **getfilep** 的 C 函数样例：

```
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 ) ;
}
```

下面是编译－生成－运行该函数的样例：

```
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%
```

有关更多信息，请参见《Fortran 编程指南》中介绍 C-Fortran 接口的章节。另请参见 **open**(2)。

1.4.21 getlog：获取用户的登录名

该子例程的调用方式如下所示：

call getlog(*name* **)**			
name	**character*** *n*	输出	用户的登录名，如果运行的进程已与终端分离，则全为空白。*n* 应足够大，以便能容纳最长的名称。

示例：**getlog**：

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

另请参见 **getlogin**(3)。

1.4.22 getpid：获取进程 ID

该函数的调用方式如下所示：

INTEGER*4 getpid			
pid = **getpid()**			
返回值	**INTEGER*4**	输出	当前进程的进程 ID。

示例：**getpid**：

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

另请参见 **getpid**(2)。

1.4.23 getuid 和 getgid：分别获取进程的用户 ID 和组 ID

getuid 和 **getgid** 分别获取进程的用户 ID 和组 ID。

1.4.23.1 getuid：获取进程的用户 ID。

该函数的调用方式如下所示：

INTEGER*4 getuid			
uid = **getuid()**			
返回值	**INTEGER*4**	输出	进程的用户 ID

1.4.23.2 getgid：获取进程的组 ID

该函数的调用方式如下所示：

INTEGER*4 getgid			
gid = getgid()			
返回值	**INTEGER*4**	输出	进程的组 ID

示例：**getuid()** 和 **getpid()**：

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

另请参见：**getuid**(2)。

1.4.24 hostnm：获取当前主机的名称

该函数的调用方式如下所示：

INTEGER*4 hostnm *status* = **hostnm(** *name* **)**			
name	**character***n*	输出	当前主机系统的名称。*n* 必须足够大，以便能容纳主机名。
返回值	**INTEGER*4**	输出	*status*=0：OK *status*>0：错误

示例：**hostnm()**：

```
      INTEGER*4 hostnm, status
      character*8 name
      status = hostnm( name )
      write(*,*) 'host name = "', name, '"'
      end
```

另请参见 **gethostname**(2)。

1.4.25 idate：返回当前日期

idate 将当前系统日期放入一个整型数组中：日期、月份和年份

该子例程的调用方式如下所示：

call idate(*iarray* **)** 标准版本			
iarray	**INTEGER*4**	输出	三元素数组：日期、月份和年份

示例：**idate**（标准版本）：

```
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 和 sigfpe：IEEE 算术

这些子程序提供了在 Fortran 程序中充分利用 ANSI/IEEE 标准 754-1985 算术所需的模式和状态。它们与函数 **ieee_flags**(3M)、**ieee_handler**(3M) 和 **sigfpe**(3) 一一对应。

概括介绍如下：

表 1–5 IEEE 算术支持例程

ieeer = ieee_flags(*action*,*mode*,*in*,*out* **)** **ieeer = ieee_handler(***action*,*exception*,*hdl* **)** **ieeer = sigfpe(** *code*, *hdl* **)**		
action	**字符**	输入
code	**sigfpe_code_type**	输入
mode	**字符**	输入
in	**字符**	输入
exception	**字符**	输入
hdl	**sigfpe_handler_type**	输入
out	**字符**	输出
返回值	**INTEGER*4**	输出

有关如何有策略地使用这些函数的详细信息，请参见 Sun 的《数值计算指南》。

如果使用 **sigfpe**，必须在浮点状态寄存器中设置对应的陷阱-启用-掩码位。SPARC 体系结构手册中介绍了详细信息。**libm** 函数 **ieee_handler** 可为您设置这些陷阱-启用-掩码位。

mode 和 *exception* 接受的字符关键字取决于 *action* 值。

表 1–6 **ieee_flags**(*action* , *mode* , *in* , *out*) 参数和操作

action = **'clearall'**	*mode*、*in* 和 *out* 未使用；返回 0

表 1–6 **ieee_flags**(*action* , *mode* , *in* , *out*) 参数和操作 （续）

action = **'clear'** 清除 *mode* ，*in* 、*out* 未使用 ； 返回 0	*mode* = **'direction'**	
	mode = **'exception'**	*in* = **'inexact'** 或 **'division'** 或 **'underflow'** 或 **'overflow'** 或 **'invalid'** 或 **'all'** 或 **'common'**
action = **'set'** 设置浮点 *mode* ，*in* 、*out* 未使用 ； 返回 0	*mode* = **'direction'**	*in* = **'nearest'** 或 **'tozero'** 或 **'positive'** 或 **'negative'**
	mode = **'exception'**	*in* = **'inexact'** 或 **'division'** 或 **'underflow'** 或 **'overflow'** 或 **'invalid'** 或 **'all'** 或 **'common'**
action = **'get'** 测试 *mode* 设置 *in* 和 *out* 可为空白或要测试的其中一个设置，返回的当前设置取决于 *mode*，也可为 '**not available**'。如果 *mode* = '**exception**'，该函数返回 0 或当前异常标志。	*mode* = **'direction'**	*out* = **'nearest'** 或 **'tozero'** 或 **'positive'** 或 **'negative'**
	mode = **'exception'**	*out* = **'inexact'** 或 **'division'** 或 **'underflow'** 或 **'overflow'** 或 **'invalid'** 或 **'all'** 或 **'common'**

表 1–7 **ieee_handler**(*action* , *in* , *out*) 参数

action = **'clear'** 清除 *in* 的用户异常处理； *out* 未使用	*in* = **'inexact'** 或 **'division'** 或 **'underflow'** 或 **'overflow'** 或 **'invalid'** 或 **'all'** 或 **'common'**
action = **'set'** 设置 *in* 的用户处理异常； *out* 是处理程序例程的地址，或者是 **floating point.h** 中定义的 **SIGFPE_DEFAULT**、**SIGFPE_ABORT** 或 **SIGFPE_IGNORE**	*in* = **'inexact'** 或 **'division'** 或 **'underflow'** 或 **'overflow'** 或 **'invalid'** 或 **'all'** 或 **'common'**

示例 1： 将舍入方向设置为向零舍入，除非硬件不支持要求的舍入模式：

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

示例 2： 将舍入方向清理为缺省方向（向最近的值舍入）：

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

示例 3： 清除所有由于产生异常而出现的位：

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

示例 4： 按如下所示检测溢出异常：

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

上述代码将 out 设置为 **overflow** 并将 **ieeer** 设置为 25（该值视平台而定）。类似的编码可检测异常，例如 **invalid** 或 **inexact**。

示例 5：**hand1.f**，编写并使用信号处理程序：

```
      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
```

将 **address** 和 **function hand** 的声明更改为 **INTEGER*8**，以便示例 5 可以在 64 位 SPARC 环境中使用 (**-m64**)。

请参见《数值计算指南》。另请参见：**floatingpoint**(3)、**signal**(3)、**sigfpe**(3)、**floatingpoint**(3F)、**ieee_flags**(3M) 和 **ieee_handler**(3M)。

1.4.26.1 floatingpoint.h：Fortran IEEE 定义

头文件 **floatingpoint.h** 定义了根据 ANSI/IEEE 标准 754-1985 实现标准浮点所使用的常量和类型。

在 Fortran 95 源程序中包含该文件的方式如下所示：

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

如果要使用该 include 文件，需在进行 Fortran 编译之前进行预处理。如果引用该 include 文件的源文件的扩展名为 **.F**、**.F90** 或 **.F95**，则会自动预处理该文件。

IEEE 舍入模式：

fp_direction_type	IEEE 舍入方向模式的类型。枚举顺序随硬件的不同而异。

SIGFPE 处理：

sigfpe_code_type	**SIGFPE** 代码的类型。
sigfpe_handler_type	处理特定的 **SIGFPE** 代码时调用的用户自定义 **SIGFPE** 异常处理程序的类型。
SIGFPE_DEFAULT	表示缺省 **SIGFPE** 异常处理的宏：得到缺省结果时，IEEE 异常继续出现；如果是其他 **SIGFPE** 代码，IEEE 异常将中止。
SIGFPE_IGNORE	表示另外一种可用的 **SIGFPE** 异常处理的宏，即忽略异常并继续执行。
SIGFPE_ABORT	表示另外一种可用的 **SIGFPE** 异常处理的宏，即中止核心转储。

IEEE 异常处理：

N_IEEE_EXCEPTION	不相同的 IEEE 浮点异常数。
fp_exception_type	**N_IEEE_EXCEPTION** 异常的类型。每个异常都指定有一个位编号。
fp_exception_field_type	专门用于至少容纳与按 **fp_exception_type** 列入的 IEEE 异常对应的 **N_IEEE_EXCEPTION** 位的类型。因此，**fp_inexact** 对应于最低有效位，**fp_invalid** 对应于第五个最低有效位。有些操作可以设置多个异常。

IEEE 分类：

fp_class_type	IEEE 浮点值和符号分类的列表。

请参见《数值计算指南》。另请参见 **ieee_environment**(3F)。

1.4.27 index、rindex 和 lnblnk：子串的索引或长度

这些函数通过字符串搜索：

index(*a1***,***a2***)**	字符串 *a1* 中第一次出现的字符串 *a2* 的索引
rindex(*a1***,***a2***)**	字符串 *a1* 中最后一次出现的字符串 *a2* 的索引
lnblnk(*a1***)**	字符串 *a1* 中最后一个非空白字符串的索引

index 有以下几种形式：

1.4.27.1 index：字符串中第一次出现某子串

索引是通过以下方式调用的内函数：

n = **index(** *a1*, *a2* **)**			
a1	**字符**	输入	主字符串
a2	**字符**	输入	子串
返回值	**INTEGER**	输出	*n*>0：*a1* 中第一次出现的 *a2* 的索引 *n*=0：*a2* 不在 *a1* 中

如果声明了 **INTEGER*8**，在针对 64 位环境进行了编译且字符变量 *a1* 是非常大的字符串（大于 2 GB）时，**index()** 将返回 **INTEGER*8** 值。

1.4.27.2 rindex：字符串中最后一次出现某子串

该函数的调用方式如下所示：

INTEGER*4 rindex n = **rindex(** *a1*, *a2* **)**			
a1	**字符**	输入	主字符串
a2	**字符**	输入	子串
返回值	**INTEGER*4 或 INTEGER*8**	输出	*n*>0：*a1* 中最后一次出现的 *a2* 的索引 *n*=0：*a2* 不在 *a1* 中，在 64 位环境中返回 **INTEGER*8** 值

1.4.27.3 lnblnk：字符串中最后一个非空白字符串

该函数的调用方式如下所示：

n = **lnblnk(** *a1* **)**			
a1	**字符**	输入	字符串
返回值	**INTEGER*4** 或 **INTEGER*8**	输出	*n*>0：*a1* 中最后一个非空白字符串的索引 *n*=0：*a1* 全部不为空白，在 64 位环境中返回 **INTEGER*8** 值

示例：**index()**、**rindex()** 和 **lnblnk()**：

```
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
```

注 – 对于编译为要在 64 位环境中运行的程序，必须将 **index**、**rindex** 和 **lnblnk**（以及它们的接收变量）声明为 **INTEGER*8**，以便处理非常大的字符串。

1.4.28 inmax：返回最大正整数

该函数的调用方式如下所示：

m = **inmax()**			
返回值	**INTEGER*4**	输出	最大正整数

示例：**inmax**：

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

另请参见 **libm_single**(3F) 和 **libm_double**(3F)。另请参见第 3 章中介绍的 FORTRAN 77 非标准内函数 **ephuge()**。

1.4.29 itime：当前时间

itime 将当前系统时间放入整型数组中：小时、分钟和秒。该子例程的调用方式如下所示：

call itime(*iarray* **)**			
iarray	**INTEGER*4**	输出	三元素数组： *iarray*(1) = 小时 *iarray*(2) = 分钟 *iarray*(3) = 秒

示例：**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
```

另请参见 **time**(3F)、**ctime**(3F) 和 **fdate**(3F)。

1.4.30 kill：向进程发送信号

该函数的调用方式如下所示：

status **= kill(** *pid*, *signum* **)**			
pid	**INTEGER*4**	输入	其中一个用户进程的进程 ID。
signum	**INTEGER*4**	输入	有效的信号编号。请参见 *signal*(3)。
返回值	**INTEGER*4**	输出	*status*=0：OK *status*>0：错误代码

示例（片段）：使用 **kill()** 发送消息：

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

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

该函数将信号 *signum* 和整型信号编号发送到进程 *pid*。有效的信号编号列在 C include 文件 **/usr/include/sys/signal.h** 中。

另请参见：**kill**(2)、**signal**(3)、**signal**(3F)、**fork**(3F) 和 **perror**(3F)。

1.4.31 link 和 symlnk：创建指向现有文件的链接

link 创建指向现有文件的链接。**symlink** 创建指向现有文件的符号链接。

该函数的调用方式如下所示：

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

INTEGER*4 symlnk

status = **symlnk(** *name1*, *name2* **)**

name1	**character***n	输入	现有文件的路径名
name2	**character***n	输入	要链接到文件 *name1* 的路径名。 *name2* 不得已存在。
返回值	**INTEGER*4**	输出	*status*=0：OK *status*>0：系统错误代码

1.4.31.1 link：创建指向现有文件的链接

示例 1：**link**：创建一个指向文件 **tlink.db.data.1** 的链接 **data1**：

```
demo% cat tlink.f
        character*34 name1/'tlink.db.data.1'/, name2/'data1'/
        integer*4 link, status
        status = link( name1, name2 )
        if ( status .ne. 0 ) stop 'link: error'
        end
demo% f95 tlink.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: : 创建指向现有文件的符号链接

示例 2 : **symlnk** : 创建一个指向文件 **tlink.db.data.1** 的符号链接 **data1** :

```
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%
```

另请参见 : **link**(2)、**symlink**(2)、**perror**(3F) 和 **unlink**(3F)。

注意 : 路径名长度不能超过 **<sys/param.h>** 中定义的 **MAXPATHLEN** 值。

1.4.32 loc : 返回对象的地址

该内函数的调用方式如下所示 :

k **= loc(** arg **)**			
arg	任意类型	输入	变量或数组
返回值	**INTEGER*4** 或 **INTEGER*8**	输出	*arg* 的地址
	如果使用 **-m64** 进行编译以在 64 位环境中运行，则返回 **INTEGER*8** 指针。参见下面的说明。		

示例 : **loc** :

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

注 – 对于编译为要在 64 位环境中运行的程序，应将接收 **loc()** 函数输出的变量声明为 **INTEGER*8**。

1.4.33 long 和 short：整型对象转换

long 和 **short** 处理 **INTEGER*4** 与 **INTEGER*2** 之间的整型对象转换，在子程序调用列表中特别有用。

1.4.33.1 long：将短整型转换为长整型

该函数的调用方式如下所示：

call *ExpecLong***(long(***int2***))**		
int2	**INTEGER*2**	输入
返回值	**INTEGER*4**	输出

1.4.33.2 short：将长整型转换为短整型

该函数为：

INTEGER*2 short **call** *ExpecShort***(short(***int4***))**		
int4	**INTEGER*4**	输入
返回值	**INTEGER*2**	输出

示例（片段）：**long()** 和 **short()**：

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

ExpecLong 是用户程序调用的某子例程，要使用 *long* (**INTEGER*4**) 整型参数。而类似的子例程 *ExpecShort* 要使用 *short* (**INTEGER*2**) 整型参数。

如果在库例程调用中使用了常量且编译代码时使用了 **-i2** 选项，**long** 很有用。

在长型对象必须作为短整型传递的这类上下文环境中，**short** 很有用。将整数传递给幅度太大的短整型虽然不会导致出现错误，但会导致出现未预料的行为。

1.4.34 longjmp 和 isetjmp：返回到由 isetjmp 设置的位置

isetjmp 为 **longjmp** 设置位置；而 **longjmp** 返回到该位置。

1.4.34.1 isetjmp：为 longjmp 设置位置

该内函数的调用方式如下所示：

ival = **isetjmp(** *env* **)**			
env	**INTEGER*4**	输出	*env* 是由 12 个元素组成的整数数组。在 64 位环境中，它必须声明为 **INTEGER*8**。
返回值	**INTEGER*4**	输出	如果显式调用 **isetjmp**，*ival* = 0 如果通过 **longjmp** 调用 **isetjmp**，*ival* ≠ 0

1.4.34.2 longjmp：返回到由 isetjmp 设置的位置

该子例程的调用方式如下所示：

call longjmp(*env*, *ival* **)**			
env	**INTEGER*4**	输入	*env* 是由 **isetjmp** 初始化并由 12 个元素组成的整型数组。在 64 位环境中，它必须声明为 **INTEGER*8**
ival	**INTEGER*4**	输出	如果显式调用 **isetjmp**，*ival* = 0 如果通过 **longjmp** 调用 **isetjmp**，*ival* ≠ 0

说明

isetjmp 和 **longjmp** 例程用于处理在程序的低级例程中遇到的错误和中断。它们属于 **f95** 内函数。

如非必要，不应使用这些例程。它们受规范约束，且不可移植。有关错误和其他详细信息，请参见 **setjmp**(3V) 手册页。

isetjmp 在 *env* 中保存堆栈环境。它还会保存寄存器环境。

longjmp 会恢复上次调用 **isetjmp** 时保存的环境，并以继续执行这种方式返回值，就好像刚执行 **isetjmp** 调用返回 *ival* 值一样。

如果未调用 **longjmp**，从 **isetjmp** 返回的整型表达式 *ival* 为零，如果调用了 **longjmp**，则返回的 ival 不为零。

示例：使用 **isetjmp** 和 **longjmp** 的代码片断：

```
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
```

限制

- 必须先调用 **isetjmp**，然后才能调用 **longjmp**。
- **isetjmp** 和 **longjmp** 的 *env* 整型数组参数长度必须至少为 12 个元素。
- 必须以常规方式或通过参数将 *env* 变量从调用 **isetjmp** 的例程传递给调用 **longjmp** 的例程。
- **longjmp** 会尝试清理堆栈。必须从较低的调用级别而不是 **isetjmp** 调用 **longjmp**。
- 不能将 **isetjmp** 作为属于过程名称的参数进行传递。

请参见 **setjmp**(3V)。

1.4.35 malloc、malloc64、realloc 和 free：分配/重新分配/解除分配内存

函数 **malloc()**、**malloc64()** 和 **realloc()** 分配内存块并返回块的起始地址。返回值可以用于设置 **INTEGER** 或 Cray 样式的 **POINTER** 变量。**realloc()** 根据新的大小重新分配现有内存块。**free()** 解除分配由 **malloc()**、**malloc64()** 或 **realloc()** 分配的内存块。

注 – 这些例程在 **f95** 中以内函数实现，而在 **f77** 中是外部函数。它们不应该出现在 Fortran 95 程序的类型声明和 **EXTERNAL** 语句中，除非您要使用自己的版本。只有 **f95** 中实现 **realloc()** 例程。

符合标准的 Fortran 95 程序应该将 **ALLOCATE** 和 **DEALLOCATE** 语句用于 **ALLOCATABLE** 数组，以便执行动态内存管理，且不能直接调用 **malloc**/**realloc**/**free**。

传统的 Fortran 77 程序可以使用 **malloc()/malloc64()** 为 Cray 样式的 **POINTER** 变量赋值，这些变量的数据表示形式与 **INTEGER** 变量的数据表示形式相同。在 **f95** 中实现了 Cray 样式的 **POINTER** 变量，以便能够从 Fortran 77 进行移植。

1.4.35.1 malloc 和 malloc64：分配内存

malloc() 函数的调用方式如下所示：

k = **malloc(** *n* **)**			
n	**INTEGER**	输入	内存的字节数
返回值	**INTEGER***(Cray* **POINTER***)*	输出	*k*>0：*k* 为分配的内存块起始位置的地址 *k*=0：错误
	如果使用 **-m64** 针对 64 位环境进行了编译，则返回 **INTEGER*8** 指针值。参见下面的说明。		

注 – 该函数在 Fortran 95 中属于内函数，而在 Fortran 77 中属于外部函数。对于编译为要在 64 位环境中运行的 Fortran 77 程序，应将 **malloc()** 函数和接收其输出的变量声明为 **INTEGER*8**。有一个函数 **malloc64**(3F)，它用于实现在 32 位环境与 64 位环境之间移植程序。

k = **malloc64(** *n* **)**			
n	**INTEGER*8**	输入	内存的字节数
返回值	**INTEGER*8** *(Cray* **POINTER***)*	输出	*k*>0：*k* 是分配的内存块起始位置的地址 *k*=0：错误

这些函数会分配一片内存区域，并返回该区域起始位置的地址。（在 64 位环境中，返回的字节地址可能超出 **INTEGER*4** 数值范围－必须将接收变量声明为 **INTEGER*8** 以免内存地址被截断。）内存区域并未以任何方式初始化，因此，不应假设它已预设为什么内容，尤其是零！

示例：使用 **malloc()** 的代码片段：

```
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
```

在上面的示例中，获取了 4,000 字节的内存（**p2X** 指向该内存），并将其初始化为零。

1.4.35.2 realloc：重新分配内存

f95 内函数 **realloc()** 的调用方式如下所示：

k = **realloc**(*ptr*, n)			
ptr	**INTEGER**	输入	现有内存块的指针。（上次调用 **malloc()** 或 **realloc()** 时返回的值）。
n	**INTEGER**	输入	请求的新内存块大小（以字节数表示）。
返回值	**INTEGER** *(Cray* **POINTER***)*	输出	*k*>0：*k* 是分配的新内存块起始位置的地址 *k*=0：错误
	如果使用 **-m64** 针对 64 位环境进行了编译，则返回 **INTEGER*8** 指针值。参见下面的说明。		

realloc() 函数将 **ptr** 指向的内存块的大小更改为 **n** 字节，并返回指向（可能已移动的）新内存块的指针。内存块的内容保存不变，其大小为新内存块大小和旧内存块大小中较小者。

如果 **ptr** 为零，则 **realloc()** 的行为与 **malloc()** 的行为相同，其分配大小为 **n** 字节的新内存块。

如果 **n** 为零而 **ptr** 不为零，则指向的内存块可以进一步进行分配，且仅在终止应用程序后才返回给系统。

示例：使用 **malloc()** 和 **realloc()** 以及 Cray 样式的 **POINTER** 变量：

```
      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)
      ...
```

请注意，只有 **f95** 中实现 **realloc()** 例程。

1.4.35.3 free：解除分配由 Malloc 分配的内存

该子例程的调用方式如下所示：

call free (*ptr* **)**		
ptr	*Cray* **POINTER**	输入

free 解除分配先前由 **malloc** 和 **realloc()** 分配的内存区域。此时，该内存区域返回给内存管理器，用户程序不能再使用它。

示例：**free()**：

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

1.4.36 mvbits：移动位字段

该子例程的调用方式如下所示：

call mvbits(*src*, *ini1*, *nbits*, *des*, *ini2* **)**			
src	**INTEGER*4**	输入	来源
ini1	**INTEGER*4**	输入	来源中初始位的位置
nbits	**INTEGER*4**	输入	要移动的位数
des	**INTEGER*4**	输出	目标
ini2	**INTEGER*4**	输入	目标中初始位的位置

示例：**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%
```

请注意以下事项：

- 位是从 0 到 31 依次编号，0 是最低有效位，31 是最高有效位。
- **mvbits** 只更改 *des* 位置的 *ini2* 位到 *ini2*+*nbits*-**1** 位，而不更改 *src* 位置的位。
- 限制包括：
  - *ini1* + *nbits* ≥ 32
  - *ini2* + *nbits*≤ 32

1.4.37 perror、gerror 和 ierrno：获取系统错误消息

这些例程执行下列函数：

perror	将消息打印到 Fortran 逻辑单元 0 **stderr**。
gerror	获取（上一次检测到的系统错误的）系统错误消息。
ierrno	获取上一次检测到的系统错误的错误编号。

1.4.37.1 perror：将消息打印到逻辑单元 0 stderr

该子例程的调用方式如下所示：

call perror(*string* **)**			
string	**character***n*	输入	消息。它写在上一次检测到的系统错误的标准错误消息前面。

示例 1：

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

1.4.37.2 gerror：获取上一次检测到的系统错误的消息

该子例程或函数的调用方式如下所示：

call gerror(*string* **)**			
string	**character*** *n*	输出	上一次检测到的系统错误的消息

示例 2：**gerror()** 用作子例程：

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

示例 3：**gerror()** 用作函数；未使用 *string*：

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

1.4.37.3 ierrno：获取上一次检测到的系统错误的编号

该函数的调用方式如下所示：

n **= ierrno()**			
返回值	**INTEGER*4**	输出	上一次检测到的系统错误的编号

该数值只有在真正出现错误时才更新。可能会生成此类错误的大多数例程和 I/O 语句在调用之后会返回错误代码；该值能够比较可靠地反映导致出现错误状况的原因。

示例 4：**ierrno()**：

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

另请参见 **intro**(2) 和 **perror**(3)。

注意：

- **perror** 调用中的 *string* 的长度不能超过 127 个字符。
- **gerror** 返回的字符串长度由调用程序决定。

- 《Fortran 用户指南》中列出了 **f95** 的运行时 I/O 错误代码。

1.4.38 putc 和 fputc：向逻辑单元写入字符

putc 向逻辑单元 6 写入（通常是控制终端输出）。

fputc 向逻辑单元写入。

这些函数绕过正常的 Fortran I/O，将字符写入与 Fortran 逻辑单元关联的文件中。

请勿将正常的 Fortran 输出与相同单元中这些函数的输出混在一起。

请注意，要写入任何特殊的 \ 转义符（如换行符 **'\n'**），需要在编译时使用 FORTRAN 77 兼容选项 **-f77=backslash**。

1.4.38.1 putc：向逻辑单元 6 写入

该函数的调用方式如下所示：

INTEGER*4 putc

status **= putc(** *char* **)**

char	**字符**	输入	要写入单元的字符
返回值	**INTEGER*4**	输出	*status*=0：OK *status*>0：系统错误代码

示例：**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：向指定的逻辑单元写入

该函数的调用方式如下所示：

INTEGER*4 fputc *status* = **fputc(** *lunit*,*char* **)**			
lunit	**INTEGER*4**	输入	要写入的单元
char	**字符**	输入	要写入单元的字符
返回值	**INTEGER*4**	输出	*status*=0：OK *status*>0：系统错误代码

示例：**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%
```

另请参见 **putc**(3S)、**intro**(2) 和 **perror**(3F)。

1.4.39 qsort 和 qsort64：对一维数组的元素排序

该子例程的调用方式如下所示：

call qsort(*array*, *len*, *isize*, *compar* **)** **call qsort64(** *array*, *len8*, *isize8*, *compar* **)**			
array	**array**	输入	包含要排序的元素。
len	**INTEGER*4**	输入	数组中的元素数量。
len8	**INTEGER*8**	输入	数组中的元素数量。

isize	**INTEGER*4**	输入	元素的大小，通常： 4 代表整数或实数 8 代表双精度或复数 16 代表双复数 字符数组的字符对象长度
isize8	**INTEGER*8**	输入	元素的大小，通常： 4_8 代表整数或实数 8_8 代表双精度或复数 16_8 代表双复数 字符数组的字符对象长度
compar	函数名	输入	用户提供的 **INTEGER*2** 函数的名称。 确定排序的顺序。compar(*arg1*,*arg2*)

可在 64 位环境中将 **qsort64** 用于大于 2 GB 的数组。请确保将数组长度 *len8* 和元素大小 *isize8* 指定为 **INTEGER*8** 数据。可使用 Fortran 95 样式的常量显式指定 **INTEGER*8** 常量。

compar(*arg1*, *arg2*) 参数是 *array* 的元素，它返回以下值：

负数	如果认为 *arg1* 排在 *arg2* 前面
零	如果 *arg1* 与 *arg2* 的位置相同
正数	如果认为 *arg1* 排在 *arg2* 后面

例如：

```
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：生成介于 0 和 1 之间的随机数

反复调用 **ran** 可生成一个分布均匀的随机数序列。请参见 **lcrans**(3m)。

r = **ran**(*i*)			
i	**INTEGER*4**	输入	变量或数组元素
r	**REAL**	输出	变量或数组元素

示例：**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%
```

请注意以下事项：

- 该范围包括 0.0，但不包括 1.0。
- 该算法是一种倍增叠合型通用随机数生成器。
- 通常，在执行调用程序期间，将会设置**一次 i** 值。
- **i** 的初始值应该是较大的奇整数。
- 每次调用 **RAN** 都会获得同一序列的下一个随机数。
- 要在每次运行程序时获得不同的随机数序列，必须在每次运行时将参数设置为不同的初始值。
- **RAN** 使用该参数存储根据以下算法计算出来的下一个随机数的值：

```
SEED = 6909 * SEED + 1 (MOD 2**32)
```

- **SEED** 包含 32 位数，高 24 位转换为浮点，并返回该值。

1.4.41 rand、drand 和 irand：返回随机值

rand 返回介于 0.0 到 1.0 之间的实数值。

drand 返回介于 0.0 到 1.0 之间的双精度值。

irand 返回介于 0 到 2147483647 之间的正整数。

这些函数使用 **random**(3) 生成随机数序列。这三个函数共用同一个 256 字节的状态数组。这些函数的唯一优点是它们可以广泛用于 UNIX 系统上。要获得更好的随机数生成器，请比较 **lcrans**、**addrans** 和 **shufrans**。另请参见 **random**(3) 和《数值计算指南》。

i = **irand**(k)

r = **rand**(k)

d = **drand**(k)

k	**INTEGER*4**	输入	k=0：获取同一序列的下一个随机数 k=1：重新开始序列，返回第一个数 k>0：用作新序列的种子，返回第一个数
rand	**REAL*4**	输出	
drand	**REAL*8**	输出	
irand	**INTEGER*4**	输出	

示例：**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：重命名文件

该函数的调用方式如下所示：

INTEGER*4 rename status = rename(from, to)			
from	**character***n	输入	现有文件的路径名
to	**character***n	输入	文件的新路径名称
返回值	**INTEGER*4**	输出	*status*=0：OK *status*>0：系统错误代码

如果 *to* 指定的文件已存在，则 *from* 和 *to* 必须属于相同的文件类型，并且必须位于相同的文件系统中。如果 *to* 已存在，应先将其删除。

示例：**rename()**－将文件 **trename.old** 重命名为 **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%
```

另请参见 **rename**(2) 和 **perror**(3F)。

注意：路径名长度不能超过 **<sys/param.h>** 中定义的 **MAXPATHLEN** 值。

1.4.43 secnds：获取系统时间（秒）减去参数值后所得值

t = **secnds(** *t0* **)**			
t0	**REAL**	输入	常量、变量或数组元素
返回值	**REAL**	输出	自午夜起的秒数减去 *t0* 所得值

示例：**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%
```

请注意：

- **SECNDS** 的返回值精确到 0.01 秒。
- 该值是系统时间，即自午夜起的秒数，因此它能正确跨越午夜。
- 对于在一天快要结束时较短的时间间隔，可能会丢失一些精度。

1.4.44 set_io_err_handler 和 get_io_err_handler：设置和获取 I/O 错误处理程序

set_io_err_handler() 声明每当在指定的输入逻辑单元中检测到错误时要调用的用户自定义例程。

get_io_err_handler() 返回当前声明的错误处理例程的地址。

这些例程为模块子例程，只有当调用例程中有 **USE SUN_IO_HANDLERS** 时，才能访问这些例程。

USE SUN_IO_HANDLERS **call set_io_err_handler(***iu, subr_name, istat***)**			
iu	**INTEGER*8**	输入	逻辑单元编号
subr_name	**EXTERNAL**	输入	用户提供的错误处理程序子例程的名称。
istat	**INTEGER*4**	输出	返回状态。

USE SUN_IO_HANDLERS **call get_io_err_handler(***iu, subr_pointer, istat***)**			
iu	**INTEGER*8**	输入	逻辑单元编号
subr_pointer	**POINTER**	输出	当前声明的处理程序例程的地址。
istat	**INTEGER*4**	输出	返回状态。

SET_IO_ERR_HANDLER 设置用户提供的子例程 *subr_name*，它在出现输入错误时用作逻辑单元 *iu* 的 I/O 错误处理程序。对于格式化的文件，*iu* 必须是连接的 Fortran 逻辑单元。如果有错误，*istat* 将设置为非零值，否则设置为零。

例如，如果在打开逻辑单元 *iu* 之前调用 **SET_IO_ERR_HANDLER**，*istat* 将设置为 1001（“非法单元”）。如果 *subr_name* 为 NULL，用户错误处理将关闭，且程序恢复到缺省的 Fortran 错误处理。

可使用 **GET_IO_ERR_HANDLER** 获取当前用作相应逻辑单元的错误处理程序的函数的地址。例如，先调用 **GET_IO_ERR_HANDLER** 保存当前 I/O，然后再切换到另一个处理程序例程。以后可以通过保存的值恢复错误处理程序。

subr_name 是用户提供的例程的名称，用于处理逻辑单元 *iu* 上的 I/O 错误。运行时 I/O 库将所有相关信息传递给 *subr_name*，以使该例程可以诊断问题并在可能的情况下修复错误，然后继续运行。

用户提供的错误处理程序例程的接口如下所示：

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**			
UNIT	**INTEGER*8**	输入	报告错误的输入文件的逻辑单元编号

SRC_FILE	**CHARACTER*(*)**	输入	引起输入操作的 Fortran 源文件名称。
SRC_LINE	**INTEGER*8**	输入	输入操作的 **SRC_FILE** 中有错误的行号。
DATA_FILE	**CHARACTER*(*)**	输入	正在读取的数据文件名称。仅适用于已打开的外部文件。如果名称不可用（例如，逻辑单元 5），**DATA_FILE** 将设置为长度为零的字符数据项。
FILE_POS	**INTEGER*8**	输入	在输入文件中的当前位置（以字节数表示）。只有当 **DATA_FILE** 的名称已知时，才能定义该项。
CURR_BUFF	**CHARACTER*(*)**	输入	包含输入记录中剩余数据的字符串。错误的输入字符是字符串中的第一个字符。
CURR_ITEM	**INTEGER*8**	输入	检测到错误时，记录中已读取的输入项数（包括当前输入项）。例如：**READ(12,10)L,(ARR(I),I=1,L)** 在此示例中，如果 **CURR_ITEM** 的值为 15，表示在读取 **ARR** 的第 14 个元素时出现错误，**L** 是第一项，**ARR(1)** 是第二项，依此类推。
CORR_CHAR	**CHARACTER**	输出	由处理程序返回的用户所提供的更正字符。只有当 **CORR_ACTION** 不为零时，才使用该值。如果 **CORR_CHAR** 是无效字符，将会再次调用处理程序，直到返回有效字符。这可能会导致出现无限循环，用户要防止出现这种情形。
CORR_ACTION	**INTEGER**	输出	指定 I/O 库要采取的更正措施。如果值为零，则不采取特殊措施，库将恢复到其缺省的错误处理。如果值为 **1**，**CORR_CHAR** 将返回到 I/O 错误处理例程。

1.4.44.1 局限性

I/O 处理程序只能用另一个字符替换一次。它不能用多个字符替换一个字符。

错误恢复算法只能修复当前读取的错误字符，而不能修复在其他上下文环境中已经解释为有效字符的错误字符。例如，在进行列表控制的读取时，如果输入是 "**1.234509.8765**"，而正确的输入应该是 "**1.2345 9.8765**"，I/O 库将在第二个阶段遇到错误，因为它不是有效的数字。但此时不能回去将 '**0**' 更改为空白。

当前，这种错误处理功能不适用于名称列表控制的输入。在进行名称列表控制的输入时，如果出现错误，不会调用指定的 I/O 错误处理程序。

只能为外部文件而不能为内部文件设置 I/O 错误处理程序，这是因为没有与内部文件关联的逻辑单元。

只能针对语法错误而不能针对系统错误或语义错误（例如溢出的输入值）调用 I/O 错误处理程序。

如果用户提供的 I/O 错误处理程序不断向 I/O 库提供错误的字符，从而导致反复调用用户提供的 I/O 错误处理程序，则可能出现无限循环。如果在同一个文件位置反复出现错误，错误处理程序应该自行终止运行。一种解决方法就是通过将 **CORR_ACTION** 设置为 0 来采用缺省的错误路径。这样，I/O 库将继续运行并进行正常的错误处理。

1.4.45 sh：快速执行 sh 命令

该函数的调用方式如下所示：

INTEGER*4 sh

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

string	**character***n	输入	包含执行命令的字符串
返回值	**INTEGER*4**	输出	执行的 shell 的退出状态。有关该值的说明，请参见 *wait*(2)。

示例：**sh()**：

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

函数 **sh** 将 *string* 作为输入传递给 **sh** shell，就好像以命令方式键入该字符串。

当前进程将等到命令终止。

函数 **sh**(3f) 和 **system**(3f) 将参数字符串传递给 shell 用于执行。它们将参数字符串从 Fortran 字符值转换为 C 字符串值，并将其传递给 C 例程 **system**(3c)。例程 **sh**(3f) 和 **system**(3f) 的不同之处在于，**system** 在调用 C 例程系统前将刷新 Fortran I/O 缓冲区，而 **sh** 则不会刷新缓冲区。由于刷新缓冲区需要很长时间，因此，如果 Fortran 输出与调用结果无关，那么优先使用例程 **system**，而非例程 **sh**。

sh() 函数不能安全地用于多线程程序。请勿从多线程或并行程序中调用该函数。

另请参见：**execve**(2)、**wait**(2) 和 **system**(3c)。

注意：*string* 不能超过 1,024 个字符。

1.4.46 signal：按信号更改操作

该函数的调用方式如下所示：

INTEGER*4 signal 或 INTEGER*8 signal

n = **signal(** *signum*, *proc*, *flag* **)**

signum	**INTEGER*4**	输入	信号编号，请参见 *signal*(3)
proc	例程名称	输入	处理例程的用户信号名称；必须在外部语句中。
flag	**INTEGER*4**	输入	*flag* < 0：将 *proc* 用作信号处理程序 *flag* ≥ 0：忽略 *proc*；传递 *flag* 来指定操作： *flag* = **0**：使用缺省操作 *flag* = 1：忽略该信号
返回值	**INTEGER*4**	输出	*n*=-1：系统错误 *n*>0：上一个操作的定义 *n*>1：*n* 是本应调用的例程的地址 *n*<-1：如果 *signum* 是有效的信号编号：*n* 是本应调用的例程的地址。如果 *signum* **不是**有效的信号编号：*n* 是错误编号。
	INTEGER*8		在 64 位环境中，必须将 **signal** 和接收其输出的变量声明为 **INTEGER*8**

如果调用 *proc*，则将信号编号作为整数参数传递给它。

如果进程引发信号，缺省的操作通常是清理并中止。信号处理例程提供了捕捉特定异常或中断以便进行特殊处理的功能。

返回值可以用于后续 **signal** 调用中，以便恢复以前的操作定义。

即使没有错误，您也有可能会获得负返回值。事实上，如果将**有效的**信号编号传递给 **signal()**，获得的返回值小于 -1，这是正常的。

floatingpoint.h 定义 *proc* 值 **SIGFPE_DEFAULT**、**SIGFPE_IGNORE** 和 **SIGFPE_ABORT**。请参见第 53 页中的 “1.4.26.1 **floatingpoint.h**：Fortran IEEE 定义”。

在 64 位环境中，必须将 **signal** 和接收其输出的变量声明为 **INTEGER*8**，以免可能返回的地址被截断。

另请参见 **kill**(1)、**signal**(3) 和 **kill**(3F) 以及《数值计算指南》。

1.4.47 sleep：暂停执行一段时间

该子例程的调用方式如下所示：

call sleep(*itime* **)**			
itime	**INTEGER*4**	输入	要休止的秒数

由于系统守时粒度的影响，实际时间最多可能会比 *itime* 少 1 秒钟。

示例：**sleep()**：

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

另请参见 **sleep**(3)。

1.4.48 stat、lstat 和 fstat：获取文件状态

这些函数返回以下信息：

- 设备
- 索引节点编号
- 保护
- 硬链接数
- 用户 ID
- 组 ID
- 设备类型
- 大小
- 访问时间
- 修改时间
- 状态更改时间
- 最佳的块大小
- 分配的块

stat 和 **lstat** 都是按文件名查询。**fstat** 是按逻辑单元查询。

1.4.48.1 stat：按文件名获取文件状态

该函数的调用方式如下所示：

INTEGER*4 stat *ierr* = **stat (** *name*, *statb* **)**			
name	**character***n*	输入	文件的名称
statb	**INTEGER*4**	输出	文件的状态结构，由 13 个元素组成的数组
返回值	**INTEGER*4**	输出	*ierr*=0：OK *ierr*>0：错误代码

示例 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：按逻辑单元获取文件状态

该函数的调用方式如下所示：

INTEGER*4 fstat *ierr* = **fstat (** *lunit*, *statb* **)**			
lunit	**INTEGER*4**	输入	逻辑单元编号
statb	**INTEGER*4**	输出	文件的状态：由 13 个元素组成的数组
返回值	**INTEGER*4**	输出	*ierr*=0：OK *ierr*>0：错误代码

示例 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：按文件名获取文件状态

该函数的调用方式如下所示：

ierr = **lstat (** *name*, *statb* **)**			
name	**character***n*	输入	文件名
statb	**INTEGER*4**	输出	文件夹的状态数组，共 13 个元素
返回值	**INTEGER*4**	输出	*ierr*=0：OK *ierr*>0：错误代码

示例 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 文件状态数组的详细信息

INTEGER*4 数组 *statb* 中返回的信息的含义在 **stat**(2) 的 *stat* 结构中进行了介绍。

备用值不包括在内。顺序如下表所示：

statb(1)	索引节点所在的设备
statb(2)	相应索引节点的编号
statb(3)	保护
statb(4)	文件的硬链接数
statb(5)	属主的用户 ID
statb(6)	属主的组 ID
statb(7)	属于设备的索引节点的设备类型
statb(8)	文件的总大小
statb(9)	上次访问文件的时间
statb(10)	上次修改文件的时间
statb(11)	上次更改文件状态的时间
statb(12)	文件系统 I/O 操作的最佳块大小
statb(13)	分配的实际块数

另请参见 **stat**(2)、**access**(3F)、**perror**(3F) 和 **time**(3F)。

注意：路径名长度不能超过 **<sys/param.h>** 中定义的 **MAXPATHLEN** 值。

1.4.49 stat64、lstat64 和 fstat64：获取文件状态

它们是 **stat**、**lstat** 和 **fstat** 的 64 位“长文件”版本。除了由 13 个元素组成的数组 *statb* 必须声明为 **INTEGER*8** 之外，这些例程与非 64 位例程相同。

1.4.50 system：执行系统命令

该函数的调用方式如下所示：

INTEGER*4 system

status **= system(** *string* **)**

string	**character***n	输入	包含执行命令的字符串
返回值	**INTEGER*4**	输出	执行的 shell 的退出状态。有关该值的说明，请参见 *wait*(2)。

示例：**system()**：

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

函数 **system** 将 *string* 作为输入传递给 shell，就好像以命令方式键入该字符串。注意：*string* 不能超过 1,024 个字符。

如果 **system** 可以找到环境变量 **SHELL**，则 **system** 会将 **SHELL** 值用作命令解释程序 (shell)；否则使用 **sh**(1)。

当前进程将等到命令终止。

一直以来，对 **cc** 开发时进行了不同的假设：

- 如果 **cc** 调用 **system**，则 shell 始终为 Bourne shell。

system 函数会刷新打开的所有文件：

- 对于输出文件，缓冲区将刷新到实际文件中。
- 对于输入文件，无法预见指针的位置。

函数 **sh**(3f) 和 **system**(3f) 将参数字符串传递给 shell 用于执行。它们将参数字符串从 Fortran 字符值转换为 C 字符串值，并将其传递给 C 例程 **system**(3c)。例程 **sh**(3f) 和 **system**(3f) 的不同之处在于，**system** 在调用 C 例程 **system** 前将刷新 Fortran I/O 缓冲区，而 **sh** 则不会刷新缓冲区。由于刷新缓冲区需要很长时间，因此，如果 Fortran 输出与调用结果无关，那么优先使用例程 **system**，而非例程 **sh**。

另请参见：**execve**(2)、**wait**(2) 和 **system**(3)。

system() 函数不能安全地用于多线程程序。请勿从多线程或并行程序中调用该函数。

1.4.51 time、ctime、ltime 和 gmtime：获取系统时间

这些例程具有以下函数：

time	标准版本：获取以整数表示的系统时间（自 GMT 1970 年 1 月 1 日 0 时起至今的秒数） VMS 版本：获取以字符表示的系统时间 (hh:mm:ss)
ctime	将系统时间转换为 ASCII 字符串。
ltime	将系统时间分解成当地时间的月份、日期等等。
gmtime	将系统时间分解成 GMT 时间的月份、日期等等。

1.4.51.1 time：获取系统时间

time() 函数的调用方式如下所示：

INTEGER*4 time 或 INTEGER*8			
n = **time()** 标准版本			
返回值	**INTEGER*4**	输出	自 GMT 1970 年 1 月 1 日 0:0:0 时起至今的时间（秒）
	INTEGER*8	输出	在 64 位环境中，**time** 返回 INTEGER*8 值

函数 **time()** 返回自 GMT 1970 年 1 月 1 日 00:00:00 起至今的时间（秒）整数。这是操作系统时钟值。

示例：在操作系统中使用的 **time()** 标准版本：

```
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：将系统时间转换为字符

函数 **ctime** 转换系统时间 *stime*，并以由 24 个字符组成的 ASCII 字符串返回该值。

该函数的调用方式如下所示：

CHARACTER ctime*24			
string = **ctime(** *stime* **)**			
stime	**INTEGER*4**	输入	通过 **time()**（标准版本）获得的系统时间
返回值	**character*24**	输出	以字符串表示的系统时间。**ctime** 和 *string* 声明为 **character*24**。

下面的示例中显示了 **ctime** 返回值的格式。**ctime**(3C) 手册页中对此进行了介绍。

示例：**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：将系统时间分解成月份、日期等（当地时间）

该例程将系统时间分解成当地时区的月份、日期等。

该子例程的调用方式如下所示：

call ltime(*stime*, *tarray* **)**			
stime	**INTEGER*4**	输入	通过 **time()**（标准版本）获得的系统时间
tarray	**INTEGER*4(9)**	输出	当地系统时间，包括年份、月份、日期等

有关 **tarray** 中各元素的含义，请参见下一节。

示例：**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：将系统时间分解成月份、日期等 (GMT)

该例程将系统时间分解成 GMT 时间的月份、日期等。

此子例程的调用方式如下所示：

call gmtime(*stime*, *tarray* **)**			
stime	**INTEGER*4**	输入	通过 **time()**（标准版本）获得的系统时间
tarray	**INTEGER*4(9)**	输出	GMT 系统时间，包括年份、月份、日期等

示例：**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%
```

下面是 **ltime** 和 **gmtime** 的 **tarray()** 值：索引、单位和范围：

1	秒 (0 - 61)	6	年份－1900
2	分钟 (0 - 59)	7	星期几（星期日= 0）
3	小时 (0 - 23)	8	一年中的天数 (0 - 365)
4	一个月中的天数 (1 - 31)	9	夏令时，如果实行夏令时，则为 1。
5	自一月起的月份 (0 - 11)		

C 库例程 **ctime**(3C) 对这些值进行了定义，它解释了系统可能会返回值**大于** 59 的秒数的原因。另请参见：**idate**(3F) 和 **fdate**(3F)。

1.4.51.5 ctime64、gmtime64 和 ltime64：64 位环境的系统时间例程

这些是例程 **ctime**、**gmtime** 和 **ltime** 的对应版本，用于在 64 位环境中进行移植。除了输入变量 *stime* 必须是 **INTEGER*8** 之外，它们与这些例程相同。

在 32 位环境中使用且 *stime* 为 **INTEGER*8** 时，如果 *stime* 值超出 **INTEGER*4** 范围，**ctime64** 的返回值全是星号，而 **gmtime** 和 **ltime** 在 tarray 数组中填入 -1。

1.4.52 ttynam 和 isatty：获取终端端口的名称

ttynam 和 **isatty** 处理终端端口的名称。

1.4.52.1 ttynam：获取终端端口的名称

函数 **ttynam** 返回与逻辑单元 *lunit* 关联的终端设备的空白填充路径名。

该函数的调用方式如下所示：

CHARACTER ttynam*24 *name* **= ttynam(** *lunit* **)**			
lunit	**INTEGER*4**	输入	逻辑单元
返回值	**character***n	输出	如果返回非空白字符串：*name* 为 *lunit* 中设备的路径名。大小 ***n*** 必须足够大，以便能容纳最长的路径名。 如果返回空白字符串（全部为空白）：*lunit* 与目录 **/dev** 中的终端设备没有关联。

1.4.52.2 isatty：确定单元是否为终端

函数 **isatty** 根据逻辑单元 *lunit* 是否为终端设备返回 true 或 false。

该函数的调用方式如下所示：

terminal **= isatty(** *lunit* **)**			
lunit	**INTEGER*4**	输入	逻辑单元
返回值	**LOGICAL*4**	输出	*terminal*=true：是终端设备 *terminal*=false：不是终端设备

示例：确定 *lunit* 是否为 **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
```

输出为：

```
terminal = T, name = "/dev/ttyp1  "
```

1.4.53 unlink：删除文件

该函数的调用方式如下所示：

INTEGER*4 unlink *n* = **unlink (** *patnam* **)**			
patnam	**character***n*	输入	文件名
返回值	**INTEGER*4**	输出	*n*=0：OK *n*>0：错误

函数 **unlink** 删除由路径名 *patnam* 指定的文件。如果这是指向该文件的最后一个链接，则该文件的内容将会丢失。

示例：**unlink()** －删除文件 **tunlink.data**：

```
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
```

另请参见：**unlink**(2)、**link**(3F) 和 **perror**(3F)。注意：路径名长度不能超过 **<sys/param.h>** 中定义的 **MAXPATHLEN** 值。

1.4.54 wait：等待进程终止

该函数的调用方式如下：

INTEGER*4 wait *n* = **wait(** *status* **)**			
status	**INTEGER*4**	输出	子进程的终止状态
返回值	**INTEGER*4**	输出	*n*>0：子进程的进程 ID。 *n*<0：*n* 为系统错误代码；请参见 **wait**(2)。

wait 暂停调用程序，直到收到信号或其中一个子进程终止。如果自上一次执行 **wait** 后所有子进程都已终止，则立即返回子进程 ID。如果没有子进程，则立即返回错误代码。

示例：使用 **wait()** 的代码片段：

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

另请参见：**wait**(2)、**signal**(3F)、**kill**(3F) 和 **perror**(3F)。

◆ ◆ ◆ 第 2 章

2 Fortran 95 内函数

本章列出了 **f95** 编译器可识别的内函数名称。

2.1 标准 Fortran 95 的通用内函数

本节中介绍的 Fortran 95 通用内函数按其在 Fortran 95 标准中的功能进行分组。

所示参数是在采用关键字形式时可以用作参数关键字的名称，如 **cmplx(Y=B, KIND=M, X=A)** 中所示。

有关这些通用内过程的详细说明，请查阅 Fortran 95 标准。

2.1.1 参数存在查询函数

通用内函数名	说明
PRESENT	存在参数

2.1.2 数值函数

通用内函数名	说明
ABS (A)	绝对值
AIMAG (Z)	复数的虚部
AINT (A [, KIND])	整数截尾

通用内函数名	说明
ANINT (A [, KIND])	最近的整数
CEILING (A [, KIND])	大于或等于数值的最小整数
CMPLX (X [, Y, KIND])	转换为复数类型
CONJG (Z)	共轭复数
DBLE (A)	转换为双精度实数类型
DIM (X, Y)	正偏差
DPROD (X, Y)	双精度实数乘积
FLOOR (A [, KIND])	小于或等于数值的最大整数
INT (A [, KIND])	转换为整数类型
MAX (A1, A2 [, A3,...])	最大值
MIN (A1, A2 [, A3,...])	最小值
MOD (A, P)	余数函数
MODULO (A, P)	模数函数
NINT (A [, KIND])	最近的整数
REAL (A [, KIND])	转换为实数类型
SIGN (A, B)	符号传输

2.1.3 数学函数

通用内函数名	说明
ACOS (X)	反余弦
ASIN (X)	反正弦
ATAN (X)	反正切
ATAN2 (Y, X)	反正切
COS (X)	余弦
COSH (X)	双曲余弦
EXP (X)	指数
LOG (X)	自然对数

通用内函数名	说明
LOG10 (X)	常用对数（10 为基数）
SIN (X)	正弦
SINH (X)	双曲正弦
SQRT (X)	平方根
TAN (X)	正切
TANH (X)	双曲正切

2.1.4 字符函数

通用内函数名	说明
ACHAR (I)	按 ASCII 整理序列排列时给定位置的字符
ADJUSTL (STRING)	齐左调整
ADJUSTR (STRING)	齐右调整
CHAR (I [, KIND])	按处理器整理序列排列时给定位置的字符
IACHAR (C)	按 ASCII 整理序列排列时字符的位置
ICHAR (C)	按处理器整理序列排列时字符的位置
INDEX (STRING, SUBSTRING [, BACK])	子串的起始位置
LEN_TRIM (STRING)	长度不包含结尾的空白字符
LGE (STRING_A, STRING_B)	词法上大于或等于
LGT (STRING_A, STRING_B)	词法上大于
LLE (STRING_A, STRING_B)	词法上小于或等于
LLT (STRING_A, STRING_B)	词法上小于
REPEAT (STRING, NCOPIES)	重复并置
SCAN (STRING, SET [, BACK])	扫描字符串以查找集中的某个字符
TRIM (STRING)	删除结尾的空白字符
VERIFY (STRING, SET [, BACK])	检验字符串中的字符集

2.1.5 字符查询函数

通用内函数名	说明
LEN (STRING)	字符实体的长度

2.1.6 种类函数

通用内函数名	说明
KIND (X)	种类类型参数值
SELECTED_INT_KIND (R)	指定范围的整数种类类型参数
SELECTED_REAL_KIND ([P, R])	指定精度和范围的实数种类类型参数值

2.1.7 逻辑函数

通用内函数名	说明
LOGICAL (L [, KIND])	在种类类型参数不相同的逻辑类型对象之间转换

2.1.8 数值查询函数

通用内函数名	说明
DIGITS (X)	模型的有效数字数
EPSILON (X)	与此相比几乎可以忽略的数值
HUGE (X)	模型中最大的数值
MAXEXPONENT (X)	模型的最大指数
MINEXPONENT (X)	模型的最小指数
PRECISION (X)	十进制精度
RADIX (X)	模型的基数
RANGE (X)	十进制指数范围

通用内函数名	说明
TINY (X)	模型中最小的正数

2.1.9 位查询函数

通用内函数名	说明
BIT_SIZE (I)	模型中的位数

2.1.10 位操作函数

通用内函数名	说明
BTEST (I, POS)	位测试
IAND (I, J)	逻辑 AND
IBCLR (I, POS)	清除位
IBITS (I, POS, LEN)	提取位
IBSET (I, POS)	设置位
IEOR (I, J)	互斥 OR
IOR (I, J)	包容 OR
ISHFT (I, SHIFT)	逻辑移位
ISHFTC (I, SHIFT [, SIZE])	循环移位
NOT (I)	逻辑补充

2.1.11 传送函数

通用内函数名	说明
TRANSFER (SOURCE, MOLD [, SIZE])	处理第一个参数，就好象它与第二个参数属于同一种类型

2.1.12 浮点处理函数

通用内函数名	说明
`EXPONENT (X)`	型号的指数部分
`FRACTION (X)`	数值的小数部分
`NEAREST (X, S)`	指定的方向最近的不同处理器
`RRSPACING (X)`	接近指定数值的型号相对间隔的倒数
`SCALE (X, I)`	实数乘以基数得出整数幂
`SET_EXPONENT (X, I)`	设置数值的指数部分
`SPACING (X)`	接近指定数值的型号的绝对间隔

2.1.13 向量和矩阵乘法函数

通用内函数名	说明
`DOT_PRODUCT (VECTOR_A, VECTOR_B)`	两个一级数组的点乘积
`MATMUL (MATRIX_A, MATRIX_B)`	矩阵乘法

2.1.14 约简数组函数

通用内函数名	说明
`ALL (MASK [, DIM])`	如果所有的值为 True 则为 True
`ANY (MASK [, DIM])`	如果任意值为 True 则为 True
`COUNT (MASK [, DIM])`	数组中 True 元素数
`MAXVAL (ARRAY, DIM [, MASK])` 或 `MAXVAL (ARRAY [, MASK])`	数组中的最大值
`MINVAL (ARRAY, DIM [, MASK])` 或 `MINVAL (ARRAY [, MASK])`	数组中的最小值

通用内函数名	说明
PRODUCT (ARRAY, DIM [, MASK]) 或 **PRODUCT (ARRAY [, MASK])**	数组元素的乘积
SUM (ARRAY, DIM [, MASK]) 或 **SUM (ARRAY [, MASK])**	数组元素的求和

2.1.15 数组查询函数

通用内函数名	说明
ALLOCATED (ARRAY)	数组分配状态
LBOUND (ARRAY [, DIM])	数组的维数下界
SHAPE (SOURCE)	数组或标量的形式
SIZE (ARRAY [, DIM])	数组中的元素总数
UBOUND (ARRAY [, DIM])	数组的维数上界

2.1.16 数组构造函数

通用内函数名	说明
MERGE (TSOURCE, FSOURCE, MASK)	在屏蔽下合并
PACK (ARRAY, MASK [, VECTOR])	在屏蔽下将数组压缩为一级数组
SPREAD (SOURCE, DIM, NCOPIES)	增加维数以复制数组
UNPACK (VECTOR, MASK, FIELD)	在屏蔽下将一级数组解压缩为数组

2.1.17 数组整形函数

通用内函数名	说明
RESHAPE (SOURCE, SHAPE[, PAD, ORDER])	数组整形

2.1.18 数组处理函数

通用内函数名	说明
CSHIFT (ARRAY, SHIFT [, DIM])	循环移位
EOSHIFT (ARRAY, SHIFT [, BOUNDARY, DIM]	结束移位
TRANSPOSE (MATRIX)	调换两级数组

2.1.19 数组位置函数

通用内函数名	说明
MAXLOC (ARRAY, DIM [, MASK]) 或 **MAXLOC (ARRAY [, MASK])**	数组中最大值的位置
MINLOC (ARRAY, DIM [, MASK]) 或 **MINLOC (ARRAY [, MASK])**	数组中最小值的位置

2.1.20 指针关联状态函数

通用内函数名	说明
ASSOCIATED (POINTER [, TARGET])	关联状态查询或比较
NULL ([MOLD])	返回分离的指针

2.1.21 系统环境调节过程

通用内函数名	说明
COMMAND_ARGUMENT_COUNT ()	返回命令参数的数目
GET_COMMAND ([COMMAND, LENGTH, STATUS])	返回调用程序的整个命令
GET_COMMAND_ARGUMENT (NUMBER [, VALUE, LENGTH, STATUS])	返回一个命令参数

通用内函数名	说明
GET_ENVIRONMENT_VARIABLE (NAME [, VALUE, LENGTH, STATUS, TRIM_NAME])	获得环境变量的值。

2.1.22 内子例程

通用内函数名	说明
CPU_TIME (TIME)	获取处理器的时间
DATE_AND_TIME ([DATE, TIME, ZONE, VALUES])	获取日期和时间
MVBITS (FROM, FROMPOS, LEN, TO, TOPOS)	将位从一个整数复制到另一个整数
RANDOM_NUMBER (HARVEST)	返回伪随机数值
RANDOM_SEED ([SIZE, PUT, GET])	初始化或重新启动伪随机数据产生器
SYSTEM_CLOCK ([COUNT, COUNT_RATE, COUNT_MAX])	从系统时钟中获取数据

2.1.23 内函数的专用名称

表 2–1 Fortran 95 内函数的专用名称和通用名称

	专用名称	通用名称	参数类型
	ABS (A)	**ABS (A)**	缺省实数
	ACOS (X)	**ACOS (X)**	缺省实数
	AIMAG (Z)	**AIMAG (Z)**	缺省复数
	AINT (A)	**AINT (A)**	缺省实数
	ALOG (X)	**LOG (X)**	缺省实数
	ALOG10 (X)	**LOG10 (X)**	缺省实数

表 2–1 Fortran 95 内函数的专用名称和通用名称 （续）

	专用名称	通用名称	参数类型
#	AMAX0 (A1, A2 [, A3,...])	REAL (MAX (A1, A2 [, A3,...]))	缺省整数
#	AMAX1 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	缺省实数
#	AMIN0 (A1, A2 [, A3,...])	REAL (MIN (A1, A2 [, A3,...]))	缺省整数
#	AMIN1 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	缺省实数
	AMOD (A, P)	MOD (A, P)	缺省实数
	ANINT (A)	ANINT (A)	缺省实数
	ASIN (X)	ASIN (X)	缺省实数
	ATAN (X)	ATAN (X)	缺省实数
	ATAN2 (Y, X)	ATAN2 (Y, X)	缺省实数
	CABS (A)	ABS (A)	缺省复数
	CCOS (X)	COS (X)	缺省复数
	CEXP (X)	EXP (X)	缺省复数
#	CHAR (I)	CHAR (I)	缺省整数
	CLOG (X)	LOG (X)	缺省复数
	CONJG (Z)	CONJG (Z)	缺省复数
	COS (X)	COS (X)	缺省实数
	COSH (X)	COSH (X)	缺省实数
	CSIN (X)	SIN (X)	缺省复数
	CSQRT (X)	SQRT (X)	缺省复数
	DABS (A)	ABS (A)	双精度
	DACOS (X)	ACOS (X)	双精度
	DASIN (X)	ASIN (X)	双精度
	DATAN (X)	ATAN (X)	双精度
	DATAN2 (Y, X)	ATAN2 (Y, X)	双精度
	DCOS (X)	COS (X)	双精度
	DCOSH (X)	COSH (X)	双精度
	DDIM (X, Y)	DIM (X, Y)	双精度
	DEXP (X)	EXP (X)	双精度

表 2–1 Fortran 95 内函数的专用名称和通用名称 （续）

	专用名称	通用名称	参数类型
	DIM (X, Y)	DIM (X, Y)	缺省实数
	DINT (A)	AINT (A)	双精度
	DLOG (X)	LOG (X)	双精度
	DLOG10 (X)	LOG10 (X)	双精度
#	DMAX1 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	双精度
#	DMIN1 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	双精度
	DMOD (A, P)	MOD (A, P)	双精度
	DNINT (A)	ANINT (A)	双精度
	DPROD (X, Y)	DPROD (X, Y)	缺省实数
	DSIGN (A, B)	SIGN (A, B)	双精度
	DSIN (X)	SIN (X)	双精度
	DSINH (X)	SINH (X)	双精度
	DSQRT (X)	SQRT (X)	双精度
	DTAN (X)	TAN (X)	双精度
	DTANH (X)	TANH (X)	双精度
	EXP (X)	EXP (X)	缺省实数
#	FLOAT (A)	REAL (A)	缺省整数
	IABS (A)	ABS (A)	缺省整数
#	ICHAR (C)	ICHAR (C)	缺省字符
	IDIM (X, Y)	DIM (X, Y)	缺省整数
#	IDINT (A)	INT (A)	双精度
	IDNINT (A)	NINT (A)	双精度
#	IFIX (A)	INT (A)	缺省实数
	INDEX (STRING, SUBSTRING)	INDEX (STRING, SUBSTRING)	缺省字符
#	INT (A)	INT (A)	缺省实数
	ISIGN (A, B)	SIGN (A, B)	缺省整数
	LEN (STRING)	LEN (STRING)	缺省字符
#	LGE (STRING_A, STRING_B)	LGE (STRING_A, STRING_B)	缺省字符

表 2–1 Fortran 95 内函数的专用名称和通用名称 （续）

	专用名称	通用名称	参数类型
#	LGT (STRING_A, STRING_B)	LGT (STRING_A, STRING_B)	缺省字符
#	LLE (STRING_A, STRING_B)	LLE (STRING_A, STRING_B)	缺省字符
#	LLT (STRING_A, STRING_B)	LLT (STRING_A, STRING_B)	缺省字符
#	MAX0 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	缺省整数
#	MAX1 (A1, A2 [, A3,...])	INT (MAX (A1, A2 [, A3,...]))	缺省实数
#	MIN0 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	缺省整数
#	MIN1 (A1, A2 [, A3,...])	INT (MIN (A1, A2 [, A3,...]))	缺省实数
	MOD (A, P)	MOD (A, P)	缺省整数
	NINT (A)	NINT (A)	缺省实数
#	REAL (A)	REAL (A)	缺省整数
	SIGN (A, B)	SIGN (A, B)	缺省实数
	SIN (X)	SIN (X)	缺省实数
	SINH (X)	SINH (X)	缺省实数
#	SNGL (A)	REAL (A)	双精度
	SQRT (X)	SQRT (X)	缺省实数
	TAN (X)	TAN (X)	缺省实数
	TANH (X)	TANH (X)	缺省实数

标有 # 号的函数不能用作实际参数。“双精度”表示双精度实数。

2.2 Fortran 2003 Module Routines

Fortran 2003 标准提供了一组内模块，它们定义了支持 IEEE 算术以及与 C 语言的互操作性所需的功能。

2.2.1 IEEE 算术和异常模块

Fortran 2003 标准内模块 **IEEE_EXCEPTIONS**、**IEEE_ARITHMETIC** 和 **IEEE_FEATURES** 支持采用建议的语言标准的新功能，从而支持 IEEE 算术和 IEEE 异常处理。

草案标准定义了一组查询函数、基本函数、种类函数、基本子例程和非基本子例程。后面的表中列出了这些函数和子例程。

要访问这些函数和子例程，调用例程必须包括

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

有关详细信息，请参见 Fortran 标准 (`http://www.j3-fortran.org`)。

2.2.1.1 查询函数

模块 **IEEE_EXCEPTIONS** 包含下列查询函数。

功能	说明
IEEE_SUPPORT_FLAG(FLAG[,X])	查询处理器是否支持异常。
IEEE_SUPPORT_HALTING(FLAG)	查询处理器是否支持在出现异常后控制停止异常。

模块 **IEEE_ARITHMETIC** 包含下列查询函数。

功能	说明
IEEE_SUPPORT_DATATYPE([X])	查询处理器是否支持 IEEE 算术。
IEEE_SUPPORT_DENORMAL([X])	查询处理器是否支持非规范化的数值。
IEEE_SUPPORT_DIVIDE([X])	查询处理器是否支持按 IEEE 标准规定的精度进行除法运算。
IEEE_SUPPORT_INF([X])	查询处理器是否支持 IEEE 无穷大。
IEEE_SUPPORT_IO([X])	查询处理器是否支持在格式化输入/输出期间进行 IEEE 基本转换舍入。
IEEE_SUPPORT_NAN([X])	查询处理器是否支持 IEEE 非数值。
IEEE_SUPPORT_ROUNDING(VAL[,X])	查询处理器是否支持特定的舍入模式。
IEEE_SUPPORT_SQRT([X])	查询处理器是否支持 IEEE 平方根。
IEEE_SUPPORT_STANDARD([X])	查询处理器是否支持所有的 IEEE 功能。

2.2.1.2 基本函数

模块 **IEEE_ARITHMETIC** 包含参数为实数且满足特定条件（即下表中实数参数 **X** 和 **Y** 满足 **IEEE_SUPPORT_DATATYPE(X)** 和 **IEEE_SUPPORT_DATATYPE(Y)** 为 true）的下列基本函数。

功能	说明

IEEE_CLASS(X)	IEEE 类
IEEE_COPY_SIGN(X,Y)	IEEE 复制符号函数
IEEE_IS_FINITE(X)	确定值是否为有限值。
IEEE_IS_NAN(X)	确定值是否为 IEEE 非数值。
IEEE_IS_NORMAL(X)	确定值是否正常。
IEEE_IS_NEGATIVE(X)	确定值是否为负数。
IEEE_LOGB(X)	采用 IEEE 浮点格式的无偏指数。
IEEE_NEXT_AFTER(X,Y)	返回趋向 **Y** 的下一个可表示的 **X** 邻数。
IEEE_REM(X,Y)	IEEE REM 余数函数 **X - Y*N**，其中 **N** 是最接近 **X/Y** 精确值的整数。
IEEE_RINT(X)	根据当前的舍入模式舍入为整数值。
IEEE_SCALB(X,I)	返回 **X*2**I**
IEEE_UNORDERED(X,Y)	IEEE 无序函数。如果 **X** 或 **Y** 为 NaN，则为 true，否则为 false。
IEEE_VALUE(X,CLASS)	生成 IEEE 值。

2.2.1.3 种类函数

模块 **IEEE_ARITHMETIC** 包含以下转换函数：

功能	说明
IEEE_SELECTED_REAL_KIND([P,][R])	具有指定精度和范围的 IEEE 实数的种类类型参数值。

2.2.1.4 基本子例程

模块 **IEEE_EXCEPTIONS** 包含下列基本子例程。

子例程	说明
IEEE_GET_FLAG(FLAG,FLAG_VALUE)	获取异常标志。
IEEE_GET_HALTING_MODE(FLAG,HALTING)	获取异常的停止模式。

2.2.1.5 非基本子例程

模块 **IEEE_EXCEPTIONS** 包含下列非基本子例程。

子例程	说明
IEEE_GET_STATUS(STATUS_VALUE)	获取浮点环境的当前状态。
IEEE_SET_FLAG(FLAG,FLAG_VALUE)	设置异常标志。
IEEE_SET_HALTING_MODE(FLAG,HALTING)	控制异常持续或停止。
IEEE_SET_STATUS(STATUS_VALUE)	恢复浮点环境的状态。

模块 **IEEE_ARITHMETIC** 包含下列非基本子例程。

子例程	说明
IEEE_GET_ROUNDING_MODE(ROUND_VAL)	获取当前的 IEEE 舍入模式。
IEEE_SET_ROUNDING_MODE(ROUND_VAL)	设置当前的 IEEE 舍入模式。

2.2.2 C 绑定模块

Fortran 2003 标准提供了一种引用 C 语言过程的方式。**ISO_C_BINDING** 模块按内模块函数形式定义了三个支持过程。访问这些函数需要在调用例程中使用

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

。该模块中定义的过程如下

功能	说明
C_LOC(X)	返回参数的 C 地址
C_ASSOCIATED(C_PTR_1 [, C_PTR_2])	表示 **C_PTR_1** 的关联状态，或者表示 **C_PTR_1** 和 **C_PTR_2** 是否与同一个实体关联。
C_F_POINTER(CPTR, FPTR [, SHAPE])	将指针与 C 指针的目标关联并指定其形式。

有关 **ISO_C_BINDING** 内模块的详细信息，请参见 http://www.j3-fortran.org 上的 Fortran 2003 标准的第 15 章。

2.3 非标准 Fortran 95 内函数

下列函数在 **f95** 编译器中视为内函数，但它们不属于 Fortran 95 标准。

2.3.1 基本线性代数函数 (BLAS)

在使用 **-xknown_lib=blas** 进行编译时，编译器会将对下列例程的调用识别为内函数，并对其进行优化，然后将其链接到 Sun Performance Library 实现。编译器会忽略用户提供的这些例程版本。

表 2–2 BLAS 内函数

功能	说明
CAXPY **DAXPY** **SAXPY** **ZAXPY**	标量和向量的乘积并加上向量
CCOPY **DCOPY** **SCOPY** **ZCOPY**	复制向量
CDOTC **CDOTU** **DDOT** **SDOT** **ZDOTC** **ZDOTU**	点乘积（内部乘积）
CSCAL **DSCAL** **SSCAL** **ZSCAL**	按比例缩放向量

有关这些例程的更多信息，请参见《Sun 性能库用户指南》。

2.3.2 区间运算内函数

下表列出了在针对区间运算进行编译 (**-xia**) 时编译器可识别的内函数。有关详细信息，请参见《Fortran 95 Interval Arithmetic 编程参考》。

DINTERVAL	DIVIX	INF	INTERVAL
ISEMPTY	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	VSWID
WID			

2.3.3 其他供应商的内函数

f95 编译器可识别许多由其他供应商（包括 Cray Research, Inc.）的 Fortran 编译器定义的传统内函数。这些内函数现已过时，应避免使用。

表 2–3 Cray CF90 和其他编译器的内函数

功能	参数	说明
CLOC	([C=]c)	获取字符对象的地址
COMPL	([I=]i)	逐位补充单词。使用 NOT(i)。
COT	([X=]x)	一般余切。（另外还有：DCOT、QCOT）
CSMG	([I=]i,[J=]j,[K=]k)	有条件的标量合并
DSHIFTL	([I=]i,[J=]j,[K=]k)	将双对象 i 和 j 向左移动 k 个位
DSHIFTR	([I=]i,[J=]j,[K=]k)	将双对象 i 和 j 向右移动 k 个位
EQV	([I=]i,[J=]j)	逻辑等价。使用 IOER(i,j)。
FCD	([I=]i,[J=]j)	构造字符指针。
GETPOS	([I=]i)	获取文件位置
IBCHNG	([I=]i, [POS=]j)	更改单词中指定位的通用函数。
ISHA	([I=]i, [SHIFT=]j)	一般算术移位
ISHC	([I=]i, [SHIFT=]j)	一般循环移位
ISHL	([I=]i, [SHIFT=]j)	一般左移位
LEADZ	([I=]i)	统计前导 0 位的数量
LENGTH	([I=]i)	返回成功传送的 Cray 单词数
LOC	([I=]i)	返回变量的地址（请参见第 59 页中的 “1.4.32 **loc**：返回对象的地址”）
NEQV	([I=]i,[J=]j)	逻辑非等价。使用 IOER(i,j)。
POPCNT	([I=]i)	统计设为 1 的位数。
POPPAR	([I=]i)	计算位总体的奇偶校验
SHIFT	([I=]i,[J=]j)	循环式左移。使用 ISHFT(i,j) 或 ISHFTC(i,j,k)。
SHIFTA	([I=]i,[J=]j)	带符号扩展的算术移位。
SHIFTL	([I=]i,[J=]j)	补零式左移。使用 ISHFT(i,j) 或 ISHFTC(i,j,k)。
SHIFTR	([I=]i,[J=]j)	补零式右移。使用 ISHFT(i,j) 或 ISHFTC(i,j,k)。
TIMEF	()	返回自第一次调用后经过的时间
UNIT	([I=]i)	返回 BUFFERIN 或 BUFFEROUT 的状态
XOR	([I=]i,[J=]j)	逻辑互斥 OR。使用 IOER(i,j)。

有关 VMS Fortran 77 内函数列表，另请参见第 109 页中的 “2.3.4.2 内存函数”。

2.3.4 其他扩展

Fortran 95 编译器可识别以下其他内函数：

2.3.4.1 MPI_SIZEOF

MPI_SIZEOF(*x*, *size*, *error***)**

返回指定变量 *x* 的机器表示形式的字节数大小。如果 *x* 是数组，它返回基本元素的大小，而不是整个数组的大小。

x	输入；任意类型的变量或数组
size	输出；整数；*x* 的字节数大小
error	输出；整数；设置为如果检测到错误显示错误代码，否则为零。

2.3.4.2 内存函数

内存分配、重新分配和解除分配函数 **malloc()**、**realloc()** 和 **free()** 均以 **f95** 内函数实现。有关详细信息，请参见第 62 页中的 “1.4.35 **malloc**、**malloc64**、**realloc** 和 **free**：分配/重新分配/解除分配内存”。

◆ ◆ ◆ 第 3 章

FORTRAN 77 和 VMS 内函数

本章列出了FORTRAN 77 **f95** 接受的一系列内函数，旨在帮助将传统的 FORTRAN 77 程序迁移至 Fortran 95。

在 **f95** 中，本章列出的所有 FORTRAN 77 和 VMS 函数以及前一章列出的所有 Fortran 95 函数都识别为内函数。为了帮助从传统的 FORTRAN 77 程序迁移至 **f95**，使用 **-f77=intrinsics** 进行编译会让编译器只将 FORTRAN 77 和 VMS 函数识别为内函数，但 Fortran 95 函数不会识别为内函数。

属于 Sun 扩展的 ANSI FORTRAN 77 标准的内函数标有 ¤ 符号。使用非标准内函数和库函数的程序可能无法移植到其他平台。

内函数在接受多种数据类型的参数时，有**通用**名称和**专用**名称。通常，**通用**名称返回与参数具有相同数据类型的值。但也有一些例外，如类型转换函数（表 3–2）和查询函数（表 3–7）。这些函数也可以通过函数的某个**专用**名称进行调用，以便处理专用参数数据类型。

对于处理多个数据项的函数（例如 **sign(a1,a2)**），所有数据参数的类型必须相同。

下表按以下几方面列出 FORTRAN 77 内函数：

- 内函数－描述函数的作用
- 定义－数学定义
- 参数数量－函数接受的参数的数量
- 通用名称－函数的通用名称
- 专用名称 －函数的专用名称
- 参数类型－与每个专用名称关联的数据类型
- 函数类型－针对专用参数数据类型返回的数据类型

注 – 编译器选项 **-xtypemap** 会更改变量的缺省大小，并且对内在引用产生影响。请参见第 125 页中的 “3.4 备注”以及《Fortran 用户指南》中有关缺省大小和对齐方式的介绍。

3.1 算术和数学函数

本节详细介绍算术函数、类型转换函数、三角函数以及其他函数。“a”代表函数的单个参数，“a1”和“a2”代表两个参数函数的第一个参数和第二个参数，“ar”和“ai”代表函数的复数参数的实部和虚部。

3.1.1 算术函数

表 3–1 Fortran 77 算术函数

内函数	定义	参数 数量	通用 名称	专用 名称	参数 类型	函数类型
绝对值 **请参见注释** *(6)*。	|a| = $(ar^2+ai^2)^{1/2}$	1	**ABS**	**IABS** **ABS** **DABS** **CABS** **QABS** ¤ **ZABS** ¤ **CDABS** ¤ **CQABS** ¤	**INTEGER** **REAL** **DOUBLE** **COMPLEX** **REAL*16** **DOUBLE COMPLEX** **DOUBLE COMPLEX** **COMPLEX*32**	**INTEGER** **REAL** **DOUBLE** **REAL** **REAL*16** **DOUBLE** **DOUBLE** **REAL*16**
截断 **请参见注释** *(1)*。	int(a)	1	**AINT**	**AINT** **DINT** **QINT** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
最近的整数	如果 $a \geq 0$，则为 int(a+.5) 如果 $a < 0$，则为 int(a-.5)	1	**ANINT**	**ANINT** **DNINT** **QNINT** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**

表 3–1 Fortran 77 算术函数 （续）

内函数	定义	参数数量	通用名称	专用名称	参数类型	函数类型
最近的整数	如果 a ≥ 0，则为 int(a+.5) 如果 a < 0，则为 int(a-.5)	1	NINT	NINT IDNINT IQNINT ¤	REAL DOUBLE REAL*16	INTEGER INTEGER INTEGER
余数 **请参见注释** *(1)*。	a1-int(a1/a2)*a2	2	MOD	MOD AMOD DMOD QMOD ¤	INTEGER REAL DOUBLE REAL*16	INTEGER REAL DOUBLE REAL*16
符号传输	如果 a2 ≥ 0，则为 |a1| 如果 a2 < 0，则为 -|a1|	2	SIGN	ISIGN SIGN DSIGN QSIGN ¤	INTEGER REAL DOUBLE REAL*16	INTEGER REAL DOUBLE REAL*16
正偏差	如果 a1 > a2，则为 a1-a2 如果 a1≤ a2，则为 0	2	DIM	IDIM DIM DDIM QDIM ¤	INTEGER REAL DOUBLE REAL*16	INTEGER REAL DOUBLE REAL*16
两倍和四倍乘积	a1 * a2	2	*	DPROD QPROD ¤	REAL DOUBLE	DOUBLE REAL*16
选择最大的值	max(a1, a2, …)	≥2	MAX	MAX0 AMAX1 DMAX1 QMAX1 ¤	INTEGER REAL DOUBLE REAL*16	INTEGER REAL DOUBLE REAL*16
			AMAX0	AMAX0	INTEGER	REAL
			MAX1	MAX1	REAL	INTEGER

表 3–1 Fortran 77 算术函数 （续）

内函数	定义	参数数量	通用名称	专用名称	参数类型	函数类型
选择最小的值	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 类型转换函数

表 3–2 Fortran 77 类型转换函数

转换为	参数数量	通用名称	专用名称	参数类型	函数类型
INTEGER 请参见注释 *(1)*。	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

表 3–2 Fortran 77 类型转换函数 （续）

转换为	参数数量	通用名称	专用名称	参数类型	函数类型
REAL 请参见注释 *(2)*。	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 请参见注释 *(3)*。	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 PRECISIONDOUBLE PRECISION
			DBLEQ ¤-	REAL*16 COMPLEX*32	DOUBLE PRECISION

表 3–2 Fortran 77 类型转换函数 （续）

转换为	参数数量	通用名称	专用名称	参数类型	函数类型
REAL*16 请参见注释 *(3')*。	1	QREAL¤	QREAL ¤	INTEGER	REAL*16
		QEXT ¤	QFLOAT ¤	INTEGER	REAL*16
			-	REAL	REAL*16
			QEXT ¤	INTEGER	REAL*16
			QEXTD ¤	DOUBLE	REAL*16
			-	REAL*16	REAL*16
			-	COMPLEX	REAL*16
			-	COMPLEX*16	REAL*16
			-	COMPLEX*32	REAL*16
COMPLEX 请参见注释 *(4)* 和 *(8)*。	1 个或 2 个	CMPLX	-	INTEGER	COMPLEX
			-	REAL	COMPLEX
			-	DOUBLE	COMPLEX
			-	REAL*16	COMPLEX
			-	COMPLEX	COMPLEX
			-	COMPLEX*16	COMPLEX
			-	COMPLEX*32	COMPLEX
DOUBLE COMPLEX 请参见注释 *(8)*。	1 个或 2 个	DCMPLX@	-	INTEGER	DOUBLE COMPLEX
			-	REAL	DOUBLE COMPLEX
			-	DOUBLE	DOUBLE COMPLEX
			-	REAL*16	DOUBLE COMPLEX
			-	COMPLEX	DOUBLE COMPLEX
			-	COMPLEX*16	DOUBLE COMPLEX
			-	COMPLEX*32	DOUBLE COMPLEX

表 3–2 Fortran 77 类型转换函数 （续）

转换为	参数数量	通用名称	专用名称	参数类型	函数类型
COMPLEX*32 请参见注释 *(8)*。	1 个或 2 个	QCMPLX@	-	**INTEGER**	**COMPLEX*32**
			-	**REAL**	**COMPLEX*32**
			-	**DOUBLE**	**COMPLEX*32**
			-	**REAL*16**	**COMPLEX*32**
			-	**COMPLEX**	**COMPLEX*32**
			-	**COMPLEX*16**	**COMPLEX*32**
			-	**COMPLEX*32**	**COMPLEX*32**
INTEGER 请参见注释 *(5)*。	1	-	**ICHAR**	**CHARACTER**	**INTEGER**
		-	**IACHAR** ¤		
CHARACTER 请参见注释 *(5)*。	1	-	**CHAR**	**INTEGER**	**CHARACTER**
		-	**ACHAR** ¤		

在 ASCII 平台上（包括 Sun 系统）：

- **ACHAR** 是 **CHAR** 的非标准同义词
- **IACHAR** 是 **ICHAR** 的非标准同义词

在非 ASCII 平台上，**ACHAR** 和 **IACHAR** 专门用于提供一种直接处理 ASCII 的方法。

3.1.3 三角函数

表 3–3 Fortran 77 三角函数

内函数	定义	参数数量	通用名称	专用名称	参数类型	函数类型
正弦 请参见注释 *(7)*。	sin(a)	1	SIN	SIN DSIN QSIN ¤ CSIN ZSIN ¤ CDSIN ¤ CQSIN ¤	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32
正弦（度数） 请参见注释 *(7)*。	sin(a)	1	SIND ¤	SIND ¤ DSIND ¤ QSIND ¤	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
余弦 请参见注释 *(7)*。	cos(a)	1	COS	COS DCOS QCOS ¤ CCOS ZCOS ¤ CDCOS ¤ CQCOS ¤	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32
余弦（度数） 请参见注释 *(7)*。	cos(a)	1	COSD ¤	COSD ¤ DCOSD ¤ QCOSD ¤	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
正切 请参见注释 *(7)*。	tan(a)	1	TAN	TAN DTAN QTAN ¤	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
正切（度数） 请参见注释 *(7)*。	tan(a)	1	TAND ¤	TAND ¤ DTAND ¤ QTAND ¤	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16

表 3–3 Fortran 77 三角函数 （续）

内函数	定义	参数数量	通用名称	专用名称	参数类型	函数类型
反正弦 请参见注释 *(7)*。	arcsin(a)	1	**ASIN**	**ASIN** **DASIN** **QASIN** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
反正弦（度数） 请参见注释 *(7)*。	arcsin(a)	1	**ASIND** ¤	**ASIND** ¤ **DASIND** ¤ **QASIND** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
反余弦 请参见注释 *(7)*。	arccos(a)	1	**ACOS**	**ACOS** **DACOS** **QACOS** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
反余弦（度数） 请参见注释 *(7)*。	arccos(a)	1	**ACOSD** ¤	**ACOSD** ¤ **DACOSD** ¤ **QACOSD** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
反正切 请参见注释 *(7)*。	arctan(a)	1	**ATAN**	**ATAN** **DATAN** **QATAN** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
	arctan (a1/a2)	2	**ATAN2**	**ATAN2** **DATAN2** **QATAN2** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
反正切（度数） 请参见注释 *(7)*。	arctan(a)	1	**ATAND** ¤	**ATAND** ¤ **DATAND** ¤ **QATAND** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
	arctan (a1/a2)	2	**ATAN2D**¤	**ATAN2D** ¤ **DATAN2D** ¤ **QATAN2D** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
双曲正弦 请参见注释 *(7)*。	sinh(a)	1	**SINH**	**SINH** **DSINH** **QSINH** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**

表 3–3 Fortran 77 三角函数 （续）

内函数	定义	参数数量	通用名称	专用名称	参数类型	函数类型
双曲余弦 **请参见注释** *(7)*。	cosh(a)	1	**COSH**	**COSH** **DCOSH** **QCOSH** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**
双曲正切 **请参见注释** *(7)*。	tanh(a)	1	**TANH**	**TANH** **DTANH** **QTANH** ¤	**REAL** **DOUBLE** **REAL*16**	**REAL** **DOUBLE** **REAL*16**

3.1.4 其他数学函数

表 3–4 其他 Fortran 77 数学函数

内函数	定义	参数数量	通用名称	专用名称	参数类型	函数类型
复数的虚部 **请参见注释** *(6)*。	ai	1	**IMAG**	**AIMAG** **DIMAG** ¤ **QIMAG** ¤	**COMPLEX** **DOUBLE COMPLEX** **COMPLEX*32**	**REAL** **DOUBLE** **REAL*16**
共轭复数 **请参见注释** *(6)*。	(ar, -ai)	1	**CONJG**	**CONJG** **DCONJG** ¤ **QCONJG** ¤	**COMPLEX** **DOUBLE COMPLEX** **COMPLEX*32**	**COMPLEX** **DOUBLE COMPLEX** **COMPLEX*32**
平方根	a**(1/2)	1	**SQRT**	**SQRT** **DSQRT** **QSQRT** ¤ **CSQRT** **ZSQRT** ¤ **CDSQRT** ¤ **CQSQRT** ¤	**REAL** **DOUBLE** **REAL*16** **COMPLEX** **DOUBLE COMPLEX** **DOUBLE COMPLEX** **COMPLEX*32**	**REAL** **DOUBLE** **REAL*16** **COMPLEX** **DOUBLE COMPLEX** **DOUBLE COMPLEX** **COMPLEX*32**

表 3–4 其他 Fortran 77 数学函数 （续）

内函数	定义	参数数量	通用名称	专用名称	参数类型	函数类型
立方根 请参见注释 *(8')*。	a**(1/3)	1	CBRT	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
指数	e**a	1	EXP	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
自然对数	log(a)	1	LOG	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
常用对数	log10(a)	1	LOG10	ALOG10 DLOG10 QLOG10 ¤	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
误差函数 （请参见下面的注释）	erf(a)	1	ERF	ERF ¤ DERF ¤	REAL DOUBLE	REAL DOUBLE
误差函数	1.0 - erf(a)	1	ERFC	ERFC ¤ DERFC ¤	REAL DOUBLE	REAL DOUBLE

- 误差函数： *exp(-t*t) dt* 从 *0* 到 *a* 的 *2/sqrt(pi)* x 整数

3.2 字符函数

表 3–5 Fortran 77 字符函数

内函数	定义	参数数量	专用名称	参数类型	函数类型
转换 **请参见注释** *(5)*。	转换为字符 转换为整数 **另请参见：** 表 3–2。	1 1	**CHAR** **ACHAR** ¤ **ICHAR** **IACHAR** ¤	**INTEGER** **CHARACTER**	**CHARACTER** **INTEGER**
子串的索引	字符串 a1 中子串 a2 的位置 **请参见注释** *(10)*。	2	**INDEX**	**CHARACTER**	**INTEGER**
长度	字符实体的长度 **请参见注释** *(11)*。	1	**LEN**	**CHARACTER**	**INTEGER**
词法上大于或等于	a1 ≥ a2 **请参见注释** *(12)*。	2	**LGE**	**CHARACTER**	**LOGICAL**
词法上大于	a1 > a2 **请参见注释** *(12)*。	2	**LGT**	**CHARACTER**	**LOGICAL**
词法上小于或等于	a1≤ a2 **请参见注释** *(12)*。	2	**LLE**	**CHARACTER**	**LOGICAL**
词法上小于	a1 < a2 **请参见注释** *(12)*。	2	**LLT**	**CHARACTER**	**LOGICAL**

在 ASCII 平台上（包括 Sun 系统）：

- **ACHAR** 是 **CHAR** 的非标准同义词
- **IACHAR** 是 **ICHAR** 的非标准同义词

在非 ASCII 平台上，**ACHAR** 和 **IACHAR** 专门用于提供一种直接处理 ASCII 的方法。

3.3 其他函数

其他一些函数包括按位函数、环境查询函数以及内存分配和解除分配函数。

3.3.1 位操作

这些函数都不属于 FORTRAN 77 标准。

表 3–6 Fortran 77 按位操作函数

按位操作	参数数量	专用名称	参数类型	函数类型
补	1	**NOT**	**INTEGER**	**INTEGER**
与	22	**AND IAND**	**INTEGER**	**INTEGER**
或	22	**OR IOR**	**INTEGER**	**INTEGER**
异或	22	**XOR IEOR**	**INTEGER**	**INTEGER**
移位 **请参见注释** *(14)*。	2	**ISHFT**	**INTEGER**	**INTEGER**
左移位 **请参见注释** *(14)*。	2	**LSHIFT**	**INTEGER**	**INTEGER**
右移位 **请参见注释** *(14)*。	2	**RSHIFT**	**INTEGER**	**INTEGER**
逻辑右移位 **请参见注释** *(14)*。	2	**LRSHFT**	**INTEGER**	**INTEGER**
循环移位	3	**ISHFTC**	**INTEGER**	**INTEGER**
提取位	3	**IBITS**	**INTEGER**	**INTEGER**
设置位	2	**IBSET**	**INTEGER**	**INTEGER**
测试位	2	**BTEST**	**INTEGER**	**LOGICAL**
清除位	2	**IBCLR**	**INTEGER**	**INTEGER**

以上函数可用作内函数，也可以用作外部函数。另请参见《Fortran 库参考》手册中介绍的库位操作例程。

3.3.2 环境查询函数

这些函数都不属于 FORTRAN 77 标准。

表 3–7 Fortran 77 环境查询函数

定义	参数数量	通用名称	参数类型	函数类型
编号系统的基数	1	EPBASE	INTEGER	INTEGER
			REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
有效位数	1	EPPREC	INTEGER	INTEGER
			REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
最小指数	1	EPEMIN	REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
最大指数	1	EPEMAX	REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
最小的非零数	1	EPTINY	REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16
可表示的最大数	1	EPHUGE	INTEGER	INTEGER
			REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16
Epsilon **请参见注释** *(16)*。	1	EPMRSP	REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16

3.3.3 内存

这些函数都不属于 FORTRAN 77 标准。

表 3–8 Fortran 77 内存函数

内函数	定义	参数数量	专用名称	参数类型	函数类型
位置	地址 **请参见注释** *(17)*。	1	**LOC**	**任意**	**INTEGER*4INTEGER***
分配	分配内存并返回地址。 **请参见注释** *(17)*。	1	**MALLOC** **MALLOC64**	**INTEGER*4** **INTEGER*8**	**INTEGER** **INTEGER*8**
解除分配	解除分配由 **MALLOC** 分配的内存。**请参见注释** *(17)*。	1	**FREE**	**任意**	-
大小	返回以字节数表示的参数大小 **请参见注释** *(18)*。	1	**SIZEOF**	**任意表达式**	**INTEGER**

3.4 备注

以下备注适用于本章中的所有内函数表。

- 缩写 **DOUBLE** 代表 **DOUBLE PRECISION**。
- 采用 **INTEGER** 参数的内函数接受 **INTEGER*2**、**INTEGER*4** 或 **INTEGER*8**。
- 采用 **INTEGER** 参数的 **INTEGER** 内函数返回下面确定的 **INTEGER** 类型的值。请注意，**-xtypemap** 选项可能会更改实际参数的缺省大小：
  - **mod sign dim max min and iand or ior xor ieor**－返回值的大小是最大参数大小。
  - **abs ishft lshift rshift lrshft ibset ivclr ishftc ibits**－返回值的大小是第一个参数的大小。
  - **int epbase epprec**－返回值的大小是缺省 **INTEGER** 的大小。
  - **ephuge**－返回值的大小是缺省 **INTEGER** 的大小或参数的大小，以两者中最大的值为准。

  更改缺省数据大小的选项也改变了一些内函数的使用方式。例如，在 **-dbl** 生效时，调用带 **DOUBLE COMPLEX** 参数的 **ZCOS** 会自动变为调用 **CQCOS**，这是因为参数已经提升到 **COMPLEX*32**。以下函数也具有该功能：

  aimag alog amod cabs ccbrt ccos cdabs cdcbrt cdcos cdexp cdlog cdsin cdsqrt cexp clog csin csqrt dabs dacos dacosd dasin dasind datan datand dcbrt 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
- 以下函数允许使用整数参数或任意大小的逻辑类型：

and iand ieor iiand iieor iior inot ior jiand jieor jior jnot lrshft lshift not or rshift xor

- 所示能够返回**缺省 REAL**、**DOUBLE PRECISION**、**COMPLEX** 或 **DOUBLE COMPLEX** 值的内函数将根据某些编译选项返回主要的类型。例如，如果使用 **-xtypemap=real:64,double:64** 选项进行编译：
  - 调用 **REAL** 函数返回 **REAL*8**
  - 调用 **DOUBLE PRECISION** 函数返回 **REAL*8**
  - 调用 **COMPLEX** 函数返回 **COMPLEX*16**
  - 调用 **DOUBLE COMPLEX** 函数返回 **COMPLEX*16**

  更改缺省数据类型的数据大小的其他选项有 **–r8** 和 **–dbl**，它们也会从 **DOUBLE** 提升到 **QUAD**。与这些早期编译器选项相比，**–xtypemap=** 选项更加灵活，因此优先使用该选项。

  具有通用名称的函数返回值的类型与参数相同－类型转换函数、最近的整数函数、复数参数的绝对值以及其他函数除外。如果有多个参数，它们的类型必须相同。
- 如果函数名用作实际参数，则它必须是专用名称。
- 如果函数名用作伪参数，则它不能识别子程序中的内函数，并且根据与变量和数组相同的规则确定其数据类型。

3.4.1 有关函数的注释

各表及注释 1 至 12 以《*ANSI X3.9-1978 Programming Language FORTRAN*》中的“Table of Intrinsic Functions”为基础，并增加了 Fortran 扩展。

(1) **INT**

如果 **A** 为整数类型，则 **INT(A)** 为 **A**。

如果 **A** 为实数或双精度类型：

如果 **|A| < 1**，则 **INT(A)** 为 0；如果 **|A| ≥ 1**，则 **INT(A)** 是最大的整数，但是不超过 **A** 的幅度，并且它的符号与 **A** 的符号相同。（这样的数学整数值可能太大，无法符合计算机整数类型的要求。）

如果 **A** 为复数或双复数类型，则以上规则适用于 **A** 的实部。

如果 **A** 为实数类型，则 **IFIX(A)** 与 **INT(A)** 相同。

(2) **REAL**

如果 **A** 为实数类型，则 **REAL(A)** 为 **A**。

如果 **A** 为整数或双精度类型，则 **REAL(A)** 的 **A** 有效部分的精度与实数据具有的精度差不多。

如果 **A** 为复数类型，则 **REAL(A)** 为 **A** 的实部。

如果 **A** 为双复数类型，则 **REAL(A)** 的 **A** 实部中有效部分的精度与实数据具有的精度差不多。

(3) **DBLE**

如果 **A** 为双精度类型，则 **DBLE(A)** 为 **A**。

如果 **A** 为整数或实数类型，则 **DBLE(A)** 的 **A** 有效部分的精度与双精度数据具有的精度差不多。

如果 **A** 为复数类型，则 **DBLE(A)** 的 **A** 实部中有效部分的精度与双精度数据具有的精度差不多。

如果 **A** 为 **COMPLEX*16** 类型，则 **DBLE(A)** 为 **A** 的实部。

(3') **QREAL**

如果 **A** 为 **REAL*16** 类型，则 **QREAL(A)** 为 **A**。

如果 **A** 为整数、实数或双精度类型，则 **QREAL(A)** 的 **A** 有效部分的精度与 **REAL*16** 数据具有的精度差不多。

如果 **A** 为复数或双复数类型，则 **QREAL(A)** 的 **A** 实部中有效部分的精度与 **REAL*16** 数据具有的精度差不多。

如果 **A** 为 **COMPLEX*16** 或 **COMPLEX*32** 类型，则 **QREAL(A)** 为 **A** 的实部。

(4) CMPLX

如果 **A** 为复数类型，则 **CMPLX(A)** 为 **A**。

如果 **A** 为整数、实数或双精度类型，则 **CMPLX(A)** 为 **REAL(A) + 0i**。

如果 **A1** 和 **A2** 为整数、实数或双精度类型，则 **CMPLX(A1,A2)** 为 **REAL(A1) + REAL(A2)*i**。

如果 **A** 为双复数类型，则 **CMPLX(A)** 为 **REAL(DBLE(A)) + i*REAL(DIMAG(A))**。

如果 **CMPLX** 有两个参数，则它们的类型必须相同，可以是整数、实数或双精度类型。

如果 **CMPLX** 有一个参数，则它可以是整数、实数、双精度、复数、**COMPLEX***16 或 **COMPLEX*32** 类型。

(4') **DCMPLX**

如果 **A** 为 **COMPLEX***16 类型，则 **DCMPLX(A)** 为 **A**。

如果 **A** 为整数、实数或双精度类型，则 **DCMPLX(A)** 为 **DBLE(A) + 0i**。

如果 **A1** 和 **A2** 为整数、实数或双精度类型，则 **DCMPLX(A1,A2)** 为 **DBLE(A1) + DBLE(A2)*i**。

如果 **DCMPLX** 有两个参数，则它们的类型必须相同，可以是整数、实数或双精度类型。

如果 **DCMPLX** 有一个参数，则它可以是整数、实数、双精度、复数、**COMPLEX***16 或 **COMPLEX*32** 类型。

(5) **ICHAR**

ICHAR(A) 为 **A** 在整理序列中的位置。

第一个位置为 0，最后一个位置为 **N-1, 0≤ICHAR(A)≤N-1**，其中 **N** 是整理序列中的字符数，**A** 属于长度为 1 的字符类型。

CHAR 和 **ICHAR** 在以下几方面意义相反：

- **ICHAR(CHAR(I)) = I**，适用于 **0≤I≤N-1**
  - **CHAR(ICHAR(C)) = C**，适用于能够在处理器中表示的任何字符 **C**

(6) **COMPLEX**

COMPLEX 值表示为一对有序的实数 **(ar, ai)**，其中 **ar** 为实部，**ai** 为虚部。

(7) 弧度

所有角度都以弧度表示，除非“内函数”列包含注释“（**度数**）”。

(8) **COMPLEX** 函数

COMPLEX 类型的函数的结果是主值。

(8') **CBRT**

如果 **a** 属于 **COMPLEX** 类型，则 **CBRT** 生成结果 **COMPLEX RT1=(A, B)**，其中：**A ≥ 0.0**，而且 -60 度 ≤ **arctan (B/A) < +** 60 度。

其他两个结果的计算公式可能如下所示：

- **RT2 = RT1 * (-0.5, square_root (0.75))**
- **RT3 = RT1 * (-0.5, square_root (0.75))**

(9) 参数类型

内函数引用中所有参数的类型必须相同。

(10) **INDEX**

INDEX(X,Y) 是指 **X** 中开始出现 **Y** 的位置。也就是说，它是指字符串 **X** 中第一次出现字符串 **Y** 的起始位置。

如果 **X** 中没有 **Y**，则 **INDEX(X,Y)** 为 0。

如果 **LEN(X) < LEN(Y)**，则 **INDEX(X,Y)** 为 0。

INDEX 返回缺省的 **INTEGER*4** 数据。如果针对 64 位环境进行编译，则当结果溢出 **INTEGER*4** 数据范围时，编译器将发出警告。要在 64 位环境中使用 **INDEX**，并且字符串超出 **INTEGER*4** 限制 (2 GB)，必须将 **INDEX** 函数以及接收结果的变量声明为 **INTEGER*8**。

(11) **LEN**

LEN 返回 **CHARACTER** 参数变量的声明长度。参数的实际值无关紧要。

LEN 返回缺省的 **INTEGER*4** 数据。如果针对 64 位环境进行编译，则当结果溢出 **INTEGER*4** 数据范围时，编译器将发出警告。要在 64 位环境中使用 **LEN**，并且字符串超出 **INTEGER*4** 限制 (2 GB)，必须将 **LEN** 函数以及接收结果的变量声明为 **INTEGER*8**。

(12) 词法比较

如果 **X=Y** 或者在整理序列中 **X** 位于 **Y** 之后，则 **LGE(X, Y)** 为 true；否则为 false。

如果在整理序列中 **X** 位于 **Y** 之后，则 **LGT(X, Y)** 为 true；否则为 false。

如果 **X=Y** 或者在整理序列中 **X** 位于 **Y** 之前，则 **LLE(X, Y)** 为 true；否则为 false。

如果在整理序列中 **X** 位于 **Y** 之前，则 **LLT(X, Y)** 为 true；否则为 false。

如果 **LGE**、**LGT**、**LLE** 和 **LLT** 的操作数长度不同，则会考虑较短的操作数，就好像在右边加上了空白一样。

(13) 位函数

在 VMS Fortran 中还有其他一些按位操作，但是没有实现。

(14) 移位

LSHIFT 将 *a1* 向**左**逻辑移动 *a2* 个位（内联代码）。

LRSHFT 将 *a1* 向**右**逻辑移动 *a2* 个位（内联代码）。

RSHIFT 将 *a1* 向**右**算术移动 *a2* 个位。

ISHFT 将 *a1* 向**左**逻辑移动（如果 $a2 > 0$），或者向**右**移动（如果 $a2 < 0$）。

Fortran 中的 **LSHIFT** 和 **RSHIFT** 函数相当于 C 语言的 **<<** 和 **>>** 运算符。与 C 语言中一样，语义取决于硬件。

在移位计数超出范围时，移位函数的行为与硬件有关，并且一般无法预见。在该版本中，移位计数大于 31 将导致出现与硬件有关的行为。

(15) 环境查询

只有参数的类型有意义。

(16) 厄普西隆

Epsilon 是最小的 **e**，因此 **1.0 + e** ≠ **1.0**。

(17) **LOC**、**MALLOC** 和 **FREE**

LOC 函数返回变量或外部过程的地址。函数调用 **MALLOC(** *n* **)** 会分配至少为 *n* 个字节的块，并且返回该块的地址。

在 32 位环境中，**LOC** 返回缺省的 **INTEGER*4**，在 64 位环境中，LOC 返回缺省的 **INTEGER*8**。

MALLOC 是一种库函数，不是 FORTRAN 77 中的内函数。在 32 位环境中也返回缺省的 **INTEGER*4**，在 64 位环境中返回缺省的 **INTEGER*8**。但是，在针对 64 位环境进行编译时，必须将 **MALLOC** 显式声明为 **INTEGER*8**。

在 64 位环境中，**LOC** 或 **MALLOC** 返回的值应当存储在类型为 **POINTER**、**INTEGER*4** 或 **INTEGER*8** 的变量中。**FREE** 的参数必须是上次调用 **MALLOC** 时返回的值，因此它的数据类型应为 **POINTER**、**INTEGER*4** 或 **INTEGER*8**。

MALLOC64 始终采用 **INTEGER*8** 参数（以字节数表示的内存请求大小），并且始终返回 **INTEGER*8** 值。当编译必须在 32 位环境和 64 位环境中运行的程序时，请使用该例程，而非 **MALLOC**。必须将接收变量声明为 **POINTER** 或 **INTEGER*8**。

(18) **SIZEOF**

SIZEOF 内函数不能应用于假定大小的数组、超长的字符或者子例程调用或名称。**SIZEOF** 返回缺省的 **INTEGER*4** 数据。如果针对 64 位环境进行编译，则当结果溢出 **INTEGER*4** 数据范围时，编译器将发出警告。要在 64 位环境中使用 **SIZEOF**，并且数组超出 **INTEGER*4** 限制 (2 GB)，必须将 **SIZEOF** 函数以及接收结果的变量声明为 **INTEGER*8**。

3.5 VMS 内函数

本节列出了 **f95** 可以识别的 VMS FORTRAN 内例程。当然，它们不是标准的例程。¤

3.5.1 VMS 双精度复数

表 3–9 VMS 双精度复数函数

通用名称	专用名称	功能	参数类型	结果类型
	CDABS	绝对值	**COMPLEX*16**	**REAL*8**
	CDEXP	指数，**e**a**	**COMPLEX*16**	**COMPLEX*16**
	CDLOG	自然对数	**COMPLEX*16**	**COMPLEX*16**
	CDSQRT	平方根	**COMPLEX*16**	**COMPLEX*16**
	CDSIN	正弦	**COMPLEX*16**	**COMPLEX*16**
	CDCOS	余弦	**COMPLEX*16**	**COMPLEX*16**

表 3–9 VMS 双精度复数函数 （续）

通用名称	专用名称	功能	参数类型	结果类型
DCMPLX	DCONJG	转换为 DOUBLE COMPLEX	任意数值	COMPLEX*16
	DIMAG	复数共轭	COMPLEX*16	COMPLEX*16
	DREAL	复数的虚部	COMPLEX*16	REAL*8
		复数的实部	COMPLEX*16	REAL*8

3.5.2 VMS 度数型三角函数

表 3–10 VMS 度数型三角函数

通用名称	专用名称	功能	参数类型	结果类型
SIND	SIND	正弦	-	-
	DSIND		REAL*4	REAL*4
	QSIND		REAL*8	REAL*8
			REAL*16	REAL*16
COSD	COSD	余弦	-	-
	DCOSD		REAL*4	REAL*4
	QCOSD		REAL*8	REAL*8
			REAL*16	REAL*16
TAND	TAND	正切	-	-
	DTAND		REAL*4	REAL*4
	QTAND		REAL*8	REAL*8
			REAL*16	REAL*16
ASIND	ASIND	反正弦	-	-
	DASIND		REAL*4	REAL*4
	QASIND		REAL*8	REAL*8
			REAL*16	REAL*16
ACOSD	ACOSD	反余弦	-	-
	DACOSD		REAL*4	REAL*4
	QACOSD		REAL*8	REAL*8
			REAL*16	REAL*16

表 3–10 VMS 度数型三角函数 （续）

通用名称	专用名称	功能	参数类型	结果类型
ATAND	ATAND	反正切	-	-
	DATAND		REAL*4	REAL*4
	QATAND		REAL*8	REAL*8
			REAL*16	REAL*16
ATAN2D	ATAN2D	a1/a2 的反正切	-	-
	DATAN2D		REAL*4	REAL*4
	QATAN2D		REAL*8	REAL*8
			REAL*16	REAL*16

3.5.3 VMS 位操作

表 3–11 VMS 位操作函数

通用名称	专用名称	功能	参数类型	结果类型
IBITS	IIBITS	从 a1 中初始位 a2 提取 a3 个位	-	-
	JIBITS		INTEGER*2	INTEGER*2
	KIBITS		INTEGER*4	INTEGER*4
			INTEGER*8	INTEGER*8
ISHFT	IISHFT	将 a1 逻辑移动 a2 个位；如果 a2 是正数，则向左移动；如果 a2 是负数，则向右移动	-	-
	JISHFT		INTEGER*2	INTEGER*2
	KISHFT		INTEGER*4	INTEGER*4
			INTEGER*8	INTEGER*8
ISHFTC	IISHFTC	在 a1 中，将右边的 a3 个位循环移动 a2 个位置	-	-
	JISHFTC		INTEGER*2	INTEGER*2
			INTEGER*4	INTEGER*4
IAND	IIAND	a1、a2 的按位 AND	-	-
	JIAND		INTEGER*2	INTEGER*2
			INTEGER*4	INTEGER*4

表 3–11 VMS 位操作函数 （续）

通用名称	专用名称	功能	参数类型	结果类型
IOR	**IIOR** **JIOR** **KIOR**	**a1**、**a2** 的按位 **OR**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**
IEOR	**IIEOR** **JIEOR** **KIEOR**	**a1**、**a2** 的按位互斥 **OR**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**
NOT	**INOT** **JNOT** **KNOT**	按位补充	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**
IBSET	**IIBSET** **JIBSET** **KIBSET**	在 **a1** 中，将位 **a2** 设置为 1；返回新的 **a1**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**
BTEST	**BITEST** **BJTEST** **BKTEST**	如果 **a1** 的位 **a2** 为 1，则返回 **.TRUE.**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**	**-** **LOGICAL** **LOGICAL** **LOGICAL**
IBCLR	**IIBCLR** **JIBCLR** **KIBCLR**	在 **a1** 中，将位 **a2** 设置为 0；返回新的 **a1**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**	**-** **INTEGER*2** **INTEGER*4** **INTEGER*8**

3.5.4 VMS 多个整数类型

Fortran 标准没有解决可能出现的多个整数类型问题。编译器通过将专用的 **INTEGER**-to-**INTEGER** 函数名（**IABS** 等）视为一种特殊的通用名称，来处理出现的多个整数类型。可以使用参数类型选择相应的运行时例程名称，而程序员无法访问该名称。

VMS Fortran 采用了类似的方法，但是可以使用专用名称。

表 3–12 VMS 整数函数

专用名称	功能	参数类型	结果类型
IIABS	绝对值	INTEGER*2	INTEGER*2
JIABS		INTEGER*4	INTEGER*4
KIABS		INTEGER*8	INTEGER*8
IMAX0	最大值	INTEGER*2	INTEGER*2
JMAX0		INTEGER*4	INTEGER*4
IMIN0	最小值	INTEGER*2	INTEGER*2
JMIN0		INTEGER*4	INTEGER*4
IIDIM	正偏差	INTEGER*2	INTEGER*2
JIDIM		INTEGER*4	INTEGER*4
KIDIM		INTEGER*8	INTEGER*8
IMOD	a1/a2 的余数	INTEGER*2	INTEGER*2
JMOD		INTEGER*4	INTEGER*4
IISIGN	符号传输，|a1|* sign(a2)	INTEGER*2	INTEGER*2
JISIGN		INTEGER*4	INTEGER*4
KISIGN		INTEGER*8	INTEGER*8

索引

E

F

G

H

N

O

P

Q

R

S

T

U

V

W

X

按

贝

参

操

字

组

最

左