# G95

It's Free Crunch Time-Es el Momento Decisivo Libre

http://www.g95.org

Autor:
Andy Vaught

Traducción al español:
Víctor Manuel Gómez Traba
Francisco Javier Tsao Santín

# Características fundamentales de G95

- Compilador libre compatible con Fortran 95.
- La versión actual (Septiembre 2006) de g95 es la 0.91.
- GNU Open Source, licencia GPL.
- La operatividad de programas compilados puede ser modificada por una gran lista de variables de entorno, documentada en el propio programa compilado.
- TR15581– Argumentos ficticios (*dummy*) asignables a memoria, componentes de tipo derivado.
- Punteros de procedimientos estilo F2003, constructores de estructura, interoperabillidad
- Procedimientos y módulos intrínsecos F2003.
- Los argumentos ficticios de tipo `VALUE` en subrutinas se pasan por valor.
- Opción Comma en OPEN, READ, y WRITE para denotar punto decimal.
- Se pueden usar corchetes `[` y `]` para constructores de *arrays* (matrices/vectores
- La sentencia `IMPORT`, se usa en un cuerpo de interfaz para activar acceso a entidades de la zona de ámbito *scoping unit* de subprograma.
- `MIN()` y `MAX()` para tipos tanto de carácter como numéricos.
- `OPEN` para Entrada/Salida tipo "Transparent" o tipo *stream*.
- Compatibilidad hacia atrás con la Interfaz Binaria de Aplicación (*Application Binary Interface*, ABI) de g77.
- Disponibles enteros por defecto de 32 o 64 bits.
- Comando de Invocación `SYSTEM()`.
- Se permite código fuente con separadores de tabulación.
- Nombres simbólicos con la opción $.
- Datos Hollerith.
- Exensión DOUBLE COMPLEX.
- Longitud variable para los nombrados `COMMON`.
- Mezcla de numérico y carácter en `COMMON` y `EQUIVALENCE`.
- Tipos (*kind*)`INTEGER`: 1, 2, 4, 8.
- Tipos `LOGICAL`: 1, 2, 4, 8.
- Tipos `REAL`: 4, 8.
- `REAL(KIND=10)` para sistemas compatibles x86. 19 dígitos de precisión, rango de valor $10^{\pm 4931}$ .
- La salida de punto flotante ordenada en lista imprime el mínimo número de dígitos necesario para únicamente distinguir el número.
- Estilo de líneas de depuración (D) de VAX.
- Opción de constantes de cadenas estilo C (p.e. 'hola\nmundo').
- Descriptores de edición \ y $.
- Intrínsecos del sistema estilo VAX (SECNDS etc.)
- Biblioteca de extensiones de sistema Unix (getenv, etime, stat, etc.)
- Detección de *arrays* no-conformes o no-asignadas en tiempo de ejecución - ver Tabla IV en: `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`
- Detección de pérdidas de memoria - ver Tabla V en: `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`
- Rastreo hacia atrás de errores en tiempo de ejecución.
- La característica de compilación eficiente previene de cascadas de compilaciones reiteradas de módulos.
- Opción de compatibilidad F. Ver `http://www.fortran.com/F`. G95 puede ser construído como un compilador F.
- Característica de suspensión/recuperación de programa disponible para x86/Linux.
- El índice de bucle real o doble precisión obsoleto es BORRADO.
- La respuesta por parte del desarrollador en un informe de errores es rápida.
- Construye con las versiones GCC 4.0.3 y 4.1.1.
- Disponible para Linux/x86, PowerPC, 64-bit Opteron, 64-bit Itanium, 64-bit Alpha.
- Disponible para Windows/Cygwin, MinGW, & Interix.
- Disponible para OSX en Power Mac G4, x86-OSX.

- Disponible para FreeBSD on x86, HP-UX 11, Sparc-Solaris, x86-Solaris, OpenBSD, NetBSD, AIX, IRIX, Tru64 UNIX en Alpha.
- También disponibles versiones Fink.
- Los binarios de las versiones 'estable' y actual para la mayoría de plataformas están disponibles en `http://ftp.g95.org`.

De vez en cuando, conozco a alguien con quien he intercambiado correos electrónicos sobre g95. El comentario más frecuente que yo recibo en esas situaciones es qué trabajo extraordinario estoy haciendo solo. Yo siempre río y señalo que yo nunca hice esto solo. El número de personas que activamente ayudaron con g95 es probablemente cercano al millar o por ahí. La asunción de que la persona escribiendo el código está haciendo todo el trabajo, cuando en realidad las personas que destilan cuelgues para una docena de líneas de código están de hecho realizando un extremadamente valioso servicio, lo que frecuentemente se pasa por alto. Escribir algo tan complicado como un moderno compilador de fortran no es algo que tu hagas por ti mismo. Lo se.

Como la mayoría de las cosas, g95 nació de la frustración. Yo escribí el código de mi tesis doctoral en fortran 77 usando g77. Fortran es un maravilloso lenguaje de computación numérica– es un rápido y sucio lenguaje para gente que cuida más la respuesta que escribir el programa. El código de mi tesis tenía un montón de estructuras bastante sofisticadas en él– listas enlazadas, *octrees*, matrices vacías, soportando generación de malla para elementos finitos, resolviendo la ecuación de Poisson, expansiones multipolares, minimización de gradiente conjugado y mucha geometría computacional. Como estaba usando fortran 77, el código se acabó muy anticuado y podría haberse beneficiado inmensamente de la asignación dinámica de memoria y tipos derivados. Y mi resis fue relajándose y yo necesité un nuevo reto.

Más allá de la conveniencia de características de lenguajes más avanzados, también he estado muy inspirado por el trabajo de Bill Kahan. Con lo que me quedé después de leer la mayoría de las ponencias de Bill fue la idea de que incluso aunque los cálculos numéricos son delicados, se puede encontrar el camino para hacer cosas tales que los errores son reducidos al punto donde nadie se preocupa más de ellos. El usuario está a menudo a merced del autor de la biblioteca en este punto.

Aunque el compilador es la parte divertida, es en las bibliotecas por las que siempre me he interesado más. Las acciones del compilador son bien y muy estrictamente definidas por el standard, y es en la biblioteca en la que la innovación y experimentación puede discurrir libre. Incluso cuando estaba en un estado bastante primitivo, había ya más *tonterías* en la biblioteca comparada con otros vendedores. El archivo de núcleo de recuperación es algo que yo he querido durante años antes de realmente tomar la oportunidad de implementarla.

Ha sido muy divertido escribir g95, y miro hacia adelante para mantenerlo en las décadas que prosiguen.

Andy Vaught
Mesa, Arizona
Octubre 2006

# Licencia

G95 está licenciado bajo la Licencia Pública General de GNU (General Public License, GPL) Para todos los detalles legales, ver `http://www.gnu.org/licenses/gpl.html`.

La biblioteca de ejecución es mayoritariamente GPL y contiene una excepción a la GPL que da a los usuarios de g95 el derecho a enlazar las bibliotecas g95 a códigos no cubiertos por la GPL y a distribuir combinaciones enlazadas sin causar que los programas resultantes sean cubiertos por la GPL, o verse afectados por la GPL de algún modo.

# Notas de Instalación

Unix (Linux/OSX/Solaris/Irix/etc.):
Abrir una consola, e ir al directorio en el cual quiere instalar g95. Para descargar e instalar g95, ejecute los siguientes comandos:

```
wget -O - http://ftp.g95.org/g95-x86-linux.tgz | tar xvfz -
ln -s $PWD/g95-install/bin/i686-pc-linux-gnu-g95 /usr/bin/g95
```

Deberían estar presentes los siguientes archivos y directorios:

```
./g95-install/
./g95-install/bin/
./g95-install/bin/i686-pc-linux-gnu-g95
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/f951
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtendS.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtend.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtbeginT.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtbeginS.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtbegin.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/cc1
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/libf95.a
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/libgcc.a
./g95-install/INSTALL
./g95-install/G95Manual.pdf
```

El archivo `cc1` es un enlace simbólico a `f951` en el mismo directorio.
Cygwin:

La opción `-mno-cygwin` permite a la version para CygWin de g95 construir ejecutables que no requieren acceso al archivo cygwin1.dll para trabajar, y así pueden ser fácilmente ejecutados en otros sistemas. También los ejecutables están libres de restricciones adjuntas a la licencia GNU GPL. Para instalar una versión de Cygwin con una opción -mno-cygwin funcionando, necesitará las bibliotecas mingw instaladas, disponibles desde el sitio de Cygwin en `http://www.cygwin.com`.
Descargue el binario de `http://ftp.g95.org/g95-x86-cygwin.tgz` a su directorio raíz Cygwin (habitualmente c:\Cygwin). Empiece una sesión Cygwin, y emita estos comandos:

```
cd /
tar -xvzf g95-x86-cygwin.tgz
```

Esto instala el ejecutable g95 en la estructura de directorio `/usr/local/bin`. Advertencia: No use WinZip para extraer los archivos del tarball o los enlaces necesarios podrían no configurarse adecuadamente.
MinGW:
El binario g95 basado en MinGW para Windows puede proporcionar dos tipos de instalación. Si se encuentra MinGW, se instala dentro de la estructura de archivos de MinGW, de otra manera, se instala una versión completa independiente con los archivos de binutils de MinGW necesarios. Descargue g95 de `http://ftp.g95.org/g95-MinGW.exe`. Si tiene MinGW, instale g95 ejecutando el instalador en la raíz del directorio de MinGW. Establezca el PATH para encontrar el directorio MinGW\bin (o el g95\bin), y establezca la variable de entorno `LIBRARY_PATH` con: `SET LIBRARY_PATH =` *path-to-MinGW*`/lib`.

Nota para Usuarios de Windows XP: MinGW actualmente permite meramente 8 megabytes para la pila. Si su aplicación requiere acceso a más memoria, intente compilar con: `-Wl,--heap=0x01000000`. Use valores hexadecimales más grandes para `--heap` hasta que su porgrama se ejecute.

# Ejecutando G95

G95 determina cómo un archivo de entrada debería ser compilado basándose en su extensión. Las extensiones del nombre de archivo permitidas para archivos de código fuente de Fortran están limitadas a `.f`, `.F`, `.for`, `.FOR`, `.f90`, `.F90`, `.f95`, `.F95`, `.f03` y `.F03`. La extensión del nombre de archivo determina si los fuentes de Fortran van a ser tratatdos como forma fija, o forma libre. Se asume que los archivos terminando en `.f`, `.F`, `.for`, y `.FOR` serán fijados a la forma de fuente compatible con los archivos del viejo f77. Se asume que los archivos terminando en `.f90`, `.F90`, `.f95`, `.F95`, `.f03` y `.F03` tendrán formato libre de código fuente. Los archivos que terminan en letras mayúsculas son preprocesados con el preprocesador de C por defecto, mientras que los archivos terminando en letras minúsculas no son preprocesadas por defecto.

Las opciones básicas para compilar código fuente Fortran con g95 son:

`-c` Compila sólo, no ejecuta el enlazador.

`-v` Muestra los programas reales invocados por g95 y sus argumentos. Particularmente útil para el seguimiento de problemas de ruta de acceso.

`-o` Especifica el nombre del archivo de salida, sea un archivo objeto o el ejecutable. Se añade automáticamente una extensión `.exe` en sistemas Windows. Si no se especifica archivo de salida, elarchivo de salida por defecto se llama `a.out` en unix, o `a.exe` en sistemas Windows.

Ejemplos sencillos:

`g95 -c hello.f90`
Compila `hello.f90` a un archivo objeto llamado `hello.o`.

`g95 hello.f90`
Compila `hello.f90` y lo enlaza para producir el ejecutable `a.out` (en unix), o `a.exe` (en sistemas MS Windows).

`g95 -c h1.f90 h2.f90 h3.f90`
Compila múltiples archivos de código fuente. Si todo funciona correctamente, se crean los archivos objeto `h1.o`, `h2.o` y `h3.o`.

`g95 -o hello h1.f90 h2.f90 h3.f90`
Compila múltiples archivos de código fuente y los enlaza juntos para producir un archivo ejecutable llamado `hello` en unix, o `hello.exe` en sistemas MS Windows.

# Sinopsis de Opciones

| | |
|---|---|
| `g95 [ -c \| -S \| -E ]` | Compila y enlaza \| Produce código ensamblador \| Lista código fuente |
| `[-g] [-pg]` | Opciones de depuración |
| `[-O[`$n$`] ]` | Nivel de optimización, $n = 0, 1, 2, 3$ |
| `[-s ]` | Quita información de depuración |
| `[-W`*warn* `] [-pedantic]` | Activación/desactivación de advertencias |
| `[-I`*dir* `]` | Directorio para buscar archivos de cabeceras |
| `[-L`*dir* `]` | Directorio para buscar archivos de biblioteca |
| `[-D macro[=valor]... ]` | Definir macro |
| `[-U macro ]` | Des-definir macro |
| `[-f opción ...]` | Opciones generales de compilación |
| `[-m opción-de-máquina ...]` | Opciones específicas de máquina. Ver manual de GCC |
| `[-o archivo-de-salida ]` | Nombre de archivo de salida |
| `archivo-de-entrada` | |

# Opciones de G95

Uso: g95 [opciones] archivo...

| | |
|---|---|
| `-pass-exit-codes` | Salir con el más alto código de error de una fase. |
| `--help` | Mostrar esta información. |

| | |
|---|---|
| `--target-help` | Mostrar opciones de intérprete de comandos dependientes del objetivo. (Usar '`-v --help`' para mostrar opciones de intérprete de comandos de subprocesos). |
| `-dumpspecs` | Mostrar todas las cadenas de especificación (*spec strings*) internas. |
| `-dumpversion` | Mostrar la versión del compilador. |
| `-dumpmachine` | Mostrar el procesador objetivo del compilador. |
| `-print-search-dirs` | Mostrar los directorios en la ruta de búsqueda del compilador |
| `-print-libgcc-file-name` | Mostrar el nombre de la biblioteca compañera del compilador. |
| `-print-file-name=`*lib* | Mostrar la ruta completa a la biblioteca *lib*. |
| `-print-prog-name=`*prog* | Mostrar la ruta completa para el componente del compilador *prog*. |
| `-print-multi-directory` | Mostrar el directorio raíz para versiones de libgcc. |
| `-print-multi-lib` | Mostrar el mapeo entre las opciones del intérprete de comandos y búsqueda de directorios de múltiples bibliotecas. |
| `-print-multi-os-directory` | Mostrar la ruta relativa a las bibliotecas del Sistema Operativo. |
| `-Wa,`*opciones* | Pasar *opciones* separadas por comas en el ensamblador. |
| `-Wp,`*opciones* | Pasar *opciones* separadas por comas en el preprocesador. |
| `-Wl,`*opciones* | Pasar *opciones* separadas por comas en el enlazador. |
| `-Xassembler` *arg* | Pasar *arg* al ensamblador. |
| `-Xpreprocessor` *arg* | Pasar *arg* al preprocesador. |
| `-Xlinker` *arg* | Pasar *arg* al enlazador. |
| `-save-temps` | No borrar archivos intermedios. |
| `-pipe` | Usar tuberías más que archivos intermedios. |
| `-time` | Tiempo de ejecución de cada subproceso. No disponible en algunas plataformas (MinGW, OSX). |
| `-specs=`*archivo* | Usa las especificaciones contenidas en el *archivo* en lugar de las internas. |
| `-std=`*standard* | Asumir que los archivos de entrada de código fuente son para el *standard*. |
| `-B` *directorio* | Añadir *directorio* a las rutas de búsqueda del compilador. |
| `-b` *máquina* | Ejecutar gcc para la *máquina* objetivo, si está instalada. |
| `-V` *versión* | Ejecutar gcc versión número *versión*, si está instalado. |
| `-v` | Mostrar los programas invocados por el compilador. |
| `-M` | Producir unas líneas de dependencia de Makefile en la salida standard. |
| `-###` | Como `-v` pero las opciones son citadas y los comandos no ejecutados. |
| `-E` | Sólo preprocesa; no compilar, ensamblar o enlazar. |
| `-S` | Compilar sólo; no ensamblar ni enlazar. |
| `-c` | Compilar y ensamblar, pero no enlazar. |
| `-o` *archivo* | Nombre del *archivo* de salida. |
| `-x` *lenguaje* | Especifica el *lenguaje* de los siguientes archivos de entrada. Lenguajes permitidos incluyen: c, c++, ensamblador, ninguno; 'none' significa revertir el comportamiento por defecto de predicción del lenguaje basado en la extensión del archivo. |

Las opciones que empiezan con `-g`, `-f`, `-m`, `-O`, `-W`, o `--param` son automáticamente pasadas en varios subprocesos invocados por g95. Para pasar otras opciones en esos procesos se deben usar las opciones `-W`*letra*. Para instrucciones para informar sobre errores, por favor mirar: `http://www.g95.org`.

Por defecto, los programas compilados con g95 no tienen optimización. La *n* en `-O`*n* especifica el nivel de optimización, de 0 a 3. Cero significa ninguna optimización, y números más altos implican una optimización más agresiva. Especificando optimización se le da al compilador permiso para cambiar el código para hacerlo más rápido. Los resultados de los cálculos son a menudo afectados de formas sutiles. Usar `-O` es lo mismo que `-O1`.

Se pueden obtener aceleraciones significativas especificando al menos `-O2 -march=`*arch* donde *arch* es tu arquitectura de procesador, p.e. `pentium4`, `athlon`, `opteron`, etc. Otras opciones típicas de Fortran son `-funroll-loops`, `-fomit-frame-pointer`, `-malign-double` y `-msse2`. Para más información sobre todas las opciones de GCC disponibles cuando se compila con g95, ver: `http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc`.

# Opciones de preprocesador

G95 puede manejar archivos que contienen construcciones del preprocesador de C.

| | |
|---|---|
| `-cpp` | Forzar que los archivos de entrada sean pasados a través del preprocesador de C |
| `-no-cpp` | Evitar que los archivos de entrada sean preprocesados |
| `-D name[=value]` | Definir una macro del preprocesador |
| `-U name` | No-definir una macro del preprocesador |
| `-E` | Mostrar solo código fuente preprocesado |
| `-I` *directorio* | Añadir *directorio* para la ruta de búsqueda de los archivos de cabeceras y módulos. Los archivos son buscados en varios directorios en este orden: El directorio del archivo de código fuente principal, el directorio actual, directorios especificados por `-I`, directorios especificados en la variable de entorno `G95_INCLUDE_PATH` y finalmente en los directorios del sistema. |

# Opciones de Fortran

| | |
|---|---|
| `-Wall` | Activar la mayoría de los mensajes de advertencia. |
| `-Werror` | Convertir advertencias en errores. |
| `-Wextra` | Activar advertencias adicionales no activadas por `-Wall`. |
| `-Wglobals` | Revisión cruzada de uso y definición del procedimiento dentro del mismo archivo de código fuente. Activado por defecto, use `-Wno-globals` para deshabilitarlo. |
| `-Wimplicit-none` | Lo mismo que `-fimplicit-none`. |
| `-Wimplicit-interface` | Advertir sobre el uso de una interfaz implícita. |
| `-Wline-truncation` | Advertir sobre líneas de código fuente cortadas. |
| `-Wmissing-intent` | Advertir si faltan las especificaciones `INTENT` en los argumentos. |
| `-Wobsolescent` | Advertir sobre construcciónes obsoletas. |
| `-Wno=`*numbers* | Desactivar una lista de advertencias separadas por comas indicada por `numbers`. |
| `-Wuninitialized` | Advertir sobre variables usadas antes de ser inicializadas. Requiere `-O2`. |
| `-Wunused-internal-procs` | Advertir si un procedimiento interno nunca es usado. |
| `-Wunused-vars` | Advertir sobre variables sin usar. |
| `-Wunused-types` | Advertir sobre tipos de módulos sin usar. No es incluído por `-Wall`. |
| `-Wunset-vars` | Advertir sobre variables no declaradas. |
| `-Wunused-module-vars` | Advertir sobre variables de módulo sin usar. Útil para construir cláusulas `ONLY`. |
| `-Wunused-module-procs` | Advertir sobre procedimientos de módulo sin usar. Útil para construir cláusulas `ONLY`. |
| `-Wunused-parameter` | Advertir sobre parámetros sin usar. No es incluído por `-Wall`. |
| `-Wprecision-loss` | Advertir sobre pérdida de precisión en una conversión de tipo implícita. |
| `-fbackslash` | Interpretar backslashes(\) en constantes de caracteres como códigos de escape. Esta opción está activada por defecto. Usar `-fno-backslash` para tratar los backslashes literalmente. |
| `-fc-binding` | Imprimir prototipos de procedimientos de C a la salida standard. |
| `-fd-comment` | Hacer sentencias de líneas ejecutables D en forma fija. |
| `-fdollar-ok` | Permitir símbolo de dolar en nombres de entidad. |
| `-fendian=`*valor* | Forzar el formato endian en lecturas y escrituras sin formato. El *valor* debe ser `big` o `little`. Anula variables de entorno de ejecución. |
| `-ffixed-form` | Asumir que el archivo de código fuente es de forma fija. |
| `-ffixed-line-length-132` | Ancho de línea de 132 caracteres en modo fijo. |
| `-ffixed-line-length-80` | Ancho de línea de 80 caracteres en modo fijo. |
| `-ffree-form` | Asumir que el código fuente está en forma libre. |
| `-ffree-line-length-huge` | Permitir líneas de código fuente muy largas (10k). |

| | |
|---|---|
| `-fimplicit-none` | Especificar que no se permite tipado implícito, a menos que se anule por declaraciones explícitas`IMPLICIT`. |
| `-fintrinsic-extensions` | Activar funciones intrínsecas específicas de g95 incluso en un modo `-std=`. |
| `-fintrinsic-extensions=` | Incluir funciones intrínsecas seleccionadas incluso en un modo `-std=`. La lista es sepadada por comas y no sensible a mayúsculas/minúsculas. |
| `-fmod=`*directorio* | Poner archivos de módulo en el directorio *directory*. |
| `-fmodule-private` | Establecer a `PRIVATE` la accesibilidad por defecto de entidades de módulo. |
| `-fmultiple-save` | Permitir que el atributo `SAVE` sea especificado múltiples veces. |
| `-fone-error` | Forzar que la compilación pare después del primer error. |
| `-ftr15581` | Activar las extensiones de *arrays* asignables TR15581 incluso en los modos `-std=F` ó `-std=f95`. |
| `-std=F` | Advertir sobre características no-F. Ver `http://www.fortran.com/F`. |
| `-std=f2003` | Revisión de Fortran 2003 estricto. |
| `-std=f95` | Revisión de Fortran 95 estricto. |
| `-i4` | Establecer tipos de enteros sin especificación a tipo=4 (32 bits). |
| `-i8` | Establecer tipos de enteros sin especificación a tipo=8 (64 bits). |
| `-r8` | Establecer tipos de reales sin especificaciones de tipos a doble precisión. |
| `-d8` | Implica `-i8` y `-r8`. |

# Opciones de Generación de Código

| | |
|---|---|
| `-fbounds-check` | Revisar los límites de los *arrays* y subcadenas en tiempo de ejecución. |
| `-fcase-upper` | Hacer todos los símbolos públicos mayúsculas. |
| `-fleading-underscore` | Añadir un carácter de subrayado inicial a nombres públicos. |
| `-fonetrip` | Ejecutar bucles `DO` al menos una vez. (`FORTRAN` 66 plagado de errores). |
| `-fpack-derived` | Intentar diseñar tipos derivados lo más compactamente posible. Requiere menos memoria, pero puede ser más lento. |
| `-fqkind=`*n* | Establecer el tipo para un real con el exponente 'q' a *n*. |
| `-fsecond-underscore` | Añadiri al final un segundo carácter de subrayado en nombres que ya tienen ese carácter (por defecto). Usar `-fno-second-underscore` para suprimir. |
| `-fshort-circuit` | Provocar que los operadores `.AND.` y `.OR.` no evalúen el segundo operando si el valor de la expresión es conocido desde el primer operando. |
| `-fsloppy-char` | Suprimir errores cuando se escriben datos no-caracteres sobre descriptores de carácter, y permitir comparaciones entre variables INTEGER y CHARACTER. |
| `-fstatic` | Poner variables locales en memoria estática donde sea posible. No es lo mismo que enlazar cosas estáticamente (`-static`). |
| `-ftrace=` | `-ftrace=frame` insertará código para permitir rastreo hacia atrás en la pila en finalización anormal de programa. Esto ralentizará tu programa. Adicionalmente `-ftrace=full` permite encontrar el número de linea de excepciones aritméticas (más lento). Por defecto es `-ftrace=none`. |
| `-funderscoring` | Añadir en terminación un carácter de subrayado en nombres globales. Esta opción está activada por defecto, usar `-fno-underscoring` para suprimir. |
| `-max-frame-size=`*n* | Como será de grande, en bytes, un marco de pila (*stack frame*) simple, antes de que los *arrays* sean asignados dinámicamente. |
| `-finteger=`*n* | Inicializar variables escalares enteras sin inicializar a *n*. |
| `-flogical=`*valor* | Inicializar variables lógicas escalares sin inicializar. Los *valores* permitidos son `none`, `true` y `false`. |
| `-freal=`*valor* | Inicializar variables escalares reales y complejas sin inicializar. *valores* legales son `none`, `zero`, `nan`, `inf`, `+inf` y `-inf`. |
| `-fpointer=`*valor* | Inicializar punteros escalares. Los *valores* permitidos son `none`, `null` y `invalid`. |

| | |
|---|---|
| `-fround=`*valor* | Controlar redondeo en tiempo de compilación. *valor* puede ser `nearest`, `plus`, `minus` y `zero`. Por defecto es redondear a más cercano, `plus` es redondear a más infinito, `minus` es a menos infinito, `zero` es hacia cero. |
| `-fzero` | Inicializar tipos numéricos a cero, valores lógicos a falso y punteros a null. Las otras opciones de inicialización anulan ésta. |

# Opciones de Directorio

| | |
|---|---|
| `-I` *directorio* | Añadir *directorio* a la ruta de búsqueda de archivos de cabecera y módulos. |
| `-L`*directorio* | Añadir *directorio* a la ruta de búsqueda de bibliotecas. |
| `-fmod=`*directorio* | Poner archivos de módulos en el *directorio* |

# Variables de Entorno

El entorno de ejecución de g95 proporciona muchas opciones para modificar el comportamiento de su programa una vez que se ejecuta. Son controlables a través de variables de entorno.

Si ejecutamos un programa compilado con g95 con la opción `--g95` volcará todas estas opciones a la salida standard. Los valores de las distintas variables son siempre cadenas de caracteres, pero las cadenas de caracteres se interpretan como auténticos valores enteros o booleanos. Sólo se examina el primer carácter de un valor booleano y ha de ser '`t`', '`f`', '`y`', '`n`', '`1`' ó '`0`' (también en mayúsculas). Si algún valor es erróneo, no se informa del error y se usa el valor por defecto. Para las variables de entorno de GCC usadas por g95, como `LIBRARY_PATH`, consulte la documentación del GCC.

| | | |
|---|---|---|
| `G95_STDIN_UNIT` | Integer | Número de unidad que será pre-conectada a la entrada standard. No hará pre-conexión si es negativo, el valor por defecto es 5. |
| `G95_STDOUT_UNIT` | Integer | Número de unidad que será pre-conectada a la salida standard. No hará pre-conexión si es negativo, el valor por defecto es 6. |
| `G95_STDERR_UNIT` | Integer | Número de unidad que será pre-conectada a la salida de error standard. No hará pre-conexión si es negativo, el valor por defecto es 0. |
| `G95_USE_STDERR` | Boolean | Envía la salida de biblioteca a la salida de error standard en lugar de a la salida standard. El valor por defecto es Sí. |
| `G95_ENDIAN` | String | Formato de datos Endian a usar en la entrada/salida de datos sin formato. Los valores son `BIG`, `LITTLE` ó `NATIVE`. El valor por defecto es `NATIVE`. |
| `G95_CR` | Boolean | Escritura del carácter "Retorno de carro para registros secuenciales formateados. El valor por defecto es TRUE en sistemas no Windows/Cygwin, FALSE en caso contrario. |
| `G95_INPUT_CR` | Boolean | Trata los comandos retorno de carro-avance de línea (CR-LF) como un marcador de registro en lugar de tán solo como avance de línea. El valor por defecto es TRUE. |
| `G95_IGNORE_ENDFILE` | Boolean | Ignora los intentos de lectura pasado el registro de fin de archivo ENDFILE en el modo de acceso secuencial. El valor por defecto es FALSE. |
| `G95_TMPDIR` | String | Directorio para archivos temporales. Se impone sobre la variable de entorno `TMP`. Si `TMP` no está asignado se usa `/var/tmp`. No hay valor por defecto. |
| `G95_UNBUFFERED_ALL` | Boolean | Si es TRUE, todas las salidas son sin *buffer*. Ésto ralentizará las salidas grandes pero puede ser útil para forzar que los datos sean reflejados inmediatamente. El valor por defecto es FALSE. |
| `G95_SHOW_LOCUS` | Boolean | Si es TRUE, escribe el nombre de archivo y número de línea cuando ocurre un error en tiempo de ejecución. El valor por defecto es TRUE. |

| | | |
|---|---|---|
| G95_OPTIONAL_PLUS | Boolean | Escribe el signo opcional '+' en números donde se permita. El valor por defecto es FALSE. |
| G95_DEFAULT_RECL | Integer | Tamaño máximo por defecto del registro en archivos secuenciales. Útil sobre todo para ajustar la longitud de línea de las unidades pre-conectadas. El valor por defecto es 50000000. |
| G95_LIST_SEPARATOR | String | Separador a usar cuando se escriben listas. Puede contener cualquier número de espacios y a lo sumo una coma. El valor por defecto es un sólo espacio. |
| G95_LIST_EXP | Integer | Última potencia de 10 que no usa el formato exponencial para la salida de listas. El valor por defecto es 6. |
| G95_COMMA | Boolean | Usa una coma como el signo decimal por defecto para entrada y salida. El valor por defecto es FALSE. |
| G95_EXPAND_UNPRINTABLE | Boolean | Para salida formateada, imprime los caracteres no imprimibles con secuencua \. El valor por defecto es FALSE. |
| G95_QUIET | Boolean | Suprime el carácter de aviso sonoro (\a) de la salida formateada. El valor por defecto es FALSE. |
| G95_SYSTEM_CLOCK | Integer | Número de ticks por segundo devueltos por la función intrínseca SYSTEM_CLOCK(). 0 deshabilita el reloj. El valor por defecto es 100000. |
| G95_SEED_RNG | Boolean | Si es TRUE, renueva la semilla del generador de números aleatorios cuando el programa se ejecuta. El valor por defecto es FALSE. |
| G95_MINUS_ZERO | Boolean | Si es TRUE, escribe los valores 0 sin el signo negativo en la salida formateada (no en lista), incluso si el valor interno es negativo o menos cero. Ésta es la forma tradicional pero no standard de escribir los ceros. El valor por defecto es FALSE. |
| G95_ABORT | Boolean | Si es TRUE, realiza un volcado (*core dump*) en caso de finalización anormal del programa. Útil para la localización del problema. El valor por defecto es FALSE. |
| G95_MEM_INIT | String | Cómo inicializar la memoria asignada. El valor por defecto es NONE para no inicializar (más rápido), NAN para un valor no numérico con la mantisa 0x40f95 o un valor hexadecimal personalizado. |
| G95_MEM_SEGMENTS | Integer | Número máximo de segmentos de memoria todavía asignados a mostrar cuando termina el programa. 0 indica ninguno, un valor menor que 0 los muestra todos. El valor por defecto es 25. |
| G95_MEM_MAXALLOC | Boolean | Si es TRUE, muestra el número máximo de bytes asignados en la memoria de usuario mientras el programa de ejecuta. El valor por defecto es FALSE. |
| G95_MEM_MXFAST | Integer | Tamaño máximo de petición para manejar peticiones desde "ejecutables rápidos" (*fastbins*) . Los *fastbin* son más rápidos pero se fragmentan más fácilmente. El valor por defecto es 64 bytes. |
| G95_MEM_TRIM_THRESHOLD | Integer | Cantidad de la más alta memoria a conservar hasta que se devuelva al sistema operativo. -1 impide la devolución de memoria al sistema. Útil en programas de larga duración. El valor por defecto es 262144. |
| G95_MEM_TOP_PAD | Integer | Espacio extra para asignar cuando se obtiene memoria del sistema operativo. Puede acelerar futuras peticiones. El valor por defecto es 0. |
| G95_SIGHUP | String | Si el programa efectúa un IGNORE, ABORT, DUMP ó DUMP-QUIT en la señal SIGHUP. El valor por defecto es ABORT. Sólo en Unix. |
| G95_SIGINT | String | Si el programa efectúa un IGNORE, ABORT, DUMP ó DUMP-QUIT en la señal SIGINT. El valor por defecto es ABORT. Sólo en Unix. |

| | | |
|---|---|---|
| G95_SIGQUIT | String | Si el programa efectúa un IGNORE, ABORT, DUMP ó DUMP-QUIT en la señal SIGQUIT. El valor por defecto es ABORT. Sólo en Unix. |
| G95_CHECKPOINT | Integer | En sistemas x86 Linux, el número de segundos transcurridos entre puntos de control para archivos de volcado (*corefile dumps*). El valor de 0 significa no volcados. |
| G95_CHECKPOINT_MSG | Boolean | Si es TRUE, esribe un mensaje a la salida de error standard cuando el proceso ha sido marcado con un punto de control. El valor por defecto es TRUE. |
| G95_FPU_ROUND | String | Establece el modo de redondeo en coma flotante. Los valores pueden ser NEAREST, UP,DOWN, ZERO. El valor por defecto es NEAREST (el más próximo). |
| G95_FPU_PRECISION | String | Precisión para los resultados intermedios. El valor puede ser 24,53 y 64. El valor por defecto es 64. Sólo disponible en sistemas x86 y compatibles. |
| G95_FPU_DENORMAL | Boolean | Provoca una excepción de coma flotante cuando se encuentran números tipo 'denormal' (más pequeños que el valor más bajo que puede ser representado de forma normal en coma flotante). El valor por defecto es FALSE. |
| G95_FPU_INVALID | Boolean | Provoca una excepción de coma flotante en una operación inválida. El valor por defecto es FALSE. |
| G95_FPU_ZERODIV | Boolean | Provoca una excepción de coma flotante cuando se divide por cero. El valor por defecto es FALSE. |
| G95_FPU_OVERFLOW | Boolean | Provoca una excepción de coma flotante en caso de *overflow*. El valor por defecto es FALSE. |
| G95_FPU_UNDERFLOW | Boolean | Provoca una excepción de coma flotante en caso de *underflow*. El valor por defecto es FALSE. |
| G95_FPU_INEXACT | Boolean | Provoca una excepción de coma flotante en caso de pérdida de precisión. El valor por defecto es FALSE. |
| G95_FPU_EXCEPTIONS | Boolean | Indica si las excepciones de coma flotante ocultas deberían mostrarse después de la terminación del programa. El valor por defecto es FALSE. |
| G95_UNIT_$x$ | String | Reestablece el nombre de la unidad por defecto para la unidad $x$.El valor por defecto es fort.$x$ |
| G95_UNBUFFERED_$x$ | Boolean | Si es TRUE, la unidad $x$ funcionará sin *buffer*. El valor por defecto es FALSE. |

# Códigos de error en tiempo de ejecución

Ejecutar un programa compilado con g95 con la opción --g95 volcará la siguiente lista de códigos de error a la salida standard:

| | |
|---|---|
| -2 | Fin de registro |
| -1 | Fin de archivo |
| 0 | Retorno efectuado con éxito |
| | Códigos de error del sistema operativo (1 - 199) |
| 200 | Conflicto en las opciones de declaración |
| 201 | Opción de declaración incorrecta |
| 202 | Falta una opción de declaración |
| 203 | Archivo ya abierto en otra unidad |
| 204 | Unidad no vinculada |
| 205 | Error de FORMAT |
| 206 | ACTION especificada incorrecta |
| 207 | Lectura despues del registro ENDFILE |
| 208 | Valor incorrecto durante la lectura |

209 *overflow* numérico en la lectura
210 Fuera de memoria
211 *Array* ya asignado
212 Desasignado un puntero incorrecto
214 Registro corrupto en archivo de acceso secuencial sin formato
215 Leyendo más datos que el tamaño del registro (RECL)
216 Escribiendo más datos que el tamaño del registro (RECL)

# Características de Fortran 2003

G95 implementa varias características de Fortran 2003. Para más detalles de todas las nuevas características de Fortran 2003, vea: `http://www.kcl.ac.uk/kis/support/cit/fortran/john_reid_new_2003.pdf`.

- Están disponibles los siguientes procedimientos intrínsecos: `COMMAND_ARGUMENT_COUNT()`, `GET_COMMAND_ARGUMENT()` , `GET_COMMAND()` and `GET_ENVIRONMENT_VARIABLE()`
- Los índices de tipo Real y Double Precision en bucles tipo `DO` no están implementados en g95.
- Se pueden usar corchetes `[` y `]` de forma alternativa a los símbolos `(/` and `/)` para constructores de *arrays*.
- TR 15581 -allocatable derived types (tipos derivados asignables). Se permite el uso del atributo `ALLOCATABLE` en los argumentos ficticios, resultados de funciones, y componentes de estructuras.
- Flujo E/S - el acceso *stream* F2003 permite a un programa Fortran leer y escribir archivos binarios sin preocuparse de la estructura del registro. Clive Page ha escrito documentación sobre esta característica , disponible en: `http://www.star.le.ac.uk/~cgp/streamIO.html`.
- sentencia `IMPORT`. Se usa en el cuerpo de un interfaz para permitir el acceso a entidades del ámbito del subprograma.
- Convención europea para números reales – la etiqueta `DECIMAL='COMMA'` en las sentencias `OPEN`, `READ` y `WRITE` permite reemplazar el punto decimal en números reales con una coma.
- `MIN()` y `MAX()` funcionan con caracteres al igual que con tipos numéricos.
- La declaración del atributo `VALUE` para el argumento ficticio de un programa provoca que el argumento verdadero sea pasado por valor.
- Se admiten los constructores de estructuras del estilo de F2003 .
- Se admiten los punteros a procedimiento al estilo de F2003.
- Construcción `BIND(C)` de F2003, y módulo `ISO_C_BINDING` que provee de una más fácil interoperabiliad con C.

# Construyendo Interfaces con programas G95

Mientras que g95 produce ejecutables independentes, ocasionalmente es deseable construir un interfaz con otros programas, habitualmente en lenguaje C. La primera dificultad que tiene un programa multilenguaje son los nombres de los símbolos públicos. G95 sigue la convención de f2c de añadir un signo '_' (de subrayado) a los nombres públicos, o dos subrayados si el nombre contiene un subrayado. Las opciones `-fno-second-underscore` y `-fno-underscoring` son útiles para forzar a g95 a que produzca nombres compatibles con su compilador C. Use el programa `nm` para ver los archivos `.o` producidos por ambos compiladores. G95 produce los nombres públicos en minúscula, a no ser que se especifique `-fupper-case`, en cuyo caso todo será en mayúsculas. Los nombres de módulos se representan como *nombre-modulo*`_MP_`*nombre-entidad*.

Después de enlazar, se dan dos casos principales: Fortran llamando a subrutinas en C y C llamando a subrutinas en Fortran. Para llamar desde C a subrutinas Fortran, las subrutinas Fortran suelen llamar a subrutinas de biblioteca Fortran que esperan que la pila se inicialice de alguna manera. Para forzar una inicialización manual desde C, llame a `g95_runtime_start()` para inicialzar la biblioteca Fortran y `g95_runtime_stop()` cuando esté hecho. El prototipo de `g95_runtime_start()` es:

```
void g95_runtime_start(int argc, char *argv[]);
```

La biblioteca ha de ser capaz de procesar opciones en el intérprete de comandos. Si esto es pesado de hacer y su programa no necesita argumentos de intérprete de comandos, pase `argc=0` y `argv=NULL`. En OSX, incluya `-lSystemStubs` cuando use g95 para ejecutar el enlazador y cuando enlace archivos objeto compilados con GCC.

F2003 provee una serie de características que permiten un interfaz más fácil con C. El atributo `BIND(C)` permite que se creen símbolos de fortran que son más facilmente referenciados desde C (u otros lenguajes). Por ejemplo:

```
SUBROUTINE foo(a) BIND(C)
```

Esta forma crea un símbolo llamado `foo` sin ningún subrayado en el nombre.. Todos los caracteres se fuerzan a minúsculas. Una forma similar es:

```
SUBROUTINE foo(a) BIND(C, name='Foo1')
```

Esto provoca que el nombre del símbolo sea `Foo1`. Dentro de fortran, la subrutina todavía es referenciada por la forma habitual `foo`, `FOO` o alguna otra combinación de mayúsculas y minúsculas.

Los programas en C pasan los argumentos por valor, mientras que fortran los pasa por referencia. F2003 provee el atributo `VALUE` para espedificar que argumentos ficticios se pasen por valor. Un ejemplo podria ser:

```
SUBROUTINE foo(a)
    INTEGER, VALUE :: a
    ...
```

Una subrutina definida como ésta todavía se puede llamar desde fortran con la restricción de que los argumentos ficticios no estarían ya asociados con los argumentos verdaderos, y el hecho de cambiar un argumento ficticio ya no cambiaría el argumento verdadero.

Las variables globales pueden ser accedidas de forma similar. La siguiente subrutina escribe el valor de la variable `VAR`, que de otra forma sería inaccesible para fortran:

```
SUBROUTINE print_it
    INTEGER, BIND(C, name='VAR') :: v
    PRINT *, v
END SUBROUTINE
```

Donde fortran considera tipos que poseen distintas características o tamaños (`KIND`), C define todos como distintos tipos. Para especificar el mismo objeto, F2003 provee un módulo intrínseco `ISO_C_BINDING` que contiene las correspondencias entre los tipos de fortran y los tipos del C. Cuando es "`USE`d", son definidos los siguientes parámetros (`PARAMETER` ):

| | |
|---|---|
| `c_int` | Tipo Integer para el tipo de C `int` |
| `c_short` | Tipo Integer para el tipo de C `short` |
| `c_long` | Tipo Integer para el tipo de C `long` |
| `c_long_long` | Tipo Integer para el tipo de C `long long` |
| `c_signed_char` | Tipo Integer para el tipo de C `char` |
| `c_size_t` | Tipo Integer para el tipo de C `size_t` |
| `c_intptr_t` | Tipo Integer del mismo tamaño que los punteros del C |
| `c_float` | Tipo Real para el tipo de C `float` |
| `c_double` | Tipo Real para el tipo de C `double` |

Existen muchas otras características en `ISO_C_BINDING`. Usando este módulo, uno puede escribir un programa:

```
SUBROUTINE foo
    USE, INTRINSIC :: ISO_C_BINDING
    INTEGER(KIND=C_INT) :: int_var
    INTEGER(KIND=C_LONG_LONG) :: big_integer
    REAL(KIND=C_FLOAT) :: float_var
    ...
```

# Usando el Generador de Números Aleatorios

```
REAL INTENT(OUT):: harvest CALL random_number(harvest)
```

Devuelve un escalar tipo `REAL` o un *array* de números aleatorios tipo `REAL` en `harvest`, $0 \leq$ `harvest` $< 1$
Estableciendo la semilla del generador de números aleatorios:

```
INTEGER, OPTIONAL, INTENT(OUT) :: sz
INTEGER, OPTIONAL, INTENT(IN) :: pt(n1)
INTEGER, OPTIONAL, INTENT(OUT) :: gt(n2)
CALL random_seed(sz,pt,gt)
```

`sz` es el mínimo número de los enteros por defecto requeridos para contener el valor de la semilla; g95 devuelve cuatro. El argumento `pt` es un *array* de enteros por defecto con tamaño `n1` $\geq$ `sz`, conteniendo los valores de las semillas proporcionados por el usuario. El argumento `gt` es un *array* de los enteros por defecto de tamaño `n2` $\geq$ `sz` conteniendo la semilla actual.

Llamar a `RANDOM_SEED()` sin argumentos inicializa la semilla a un valor determinado por el reloj actual. Esto puede ser usado para generar secuencias aleatorias diferentes para cada llamada al programa. La semilla también es inicializada a un valor basado en el tiempo al inicio del programa si la variable de entorno `G95_SEED_RNG` está puesta a TRUE. Si ninguna de estas condiciones son verdaderas, `RANDOM_NUMBER()` siempre generará la misma secuencia.

El generador fundamental/subyacente es el generador xor-shift desarrollado por George Marsaglia.

# Macros Predefinidas del Preprocesador

Las macros que siempre están definidas son:

```
__G95__ 0
__G95_MINOR__ 91
__FORTRAN__ 95
__GNUC__ 4
```

Las macros condicionales son:

```
unix windows hpux linux solaris irix aix netbsd freebsd openbsd cygwin
```

# Reanudado de la ejecución de un programa mediante un archivo "Corefile"

En los sistemas Linux x86, la ejecución de un programa compilado con g95 puede ser suspendida y reanudada. Si interrumpe un programa enviándole la señal QUIT, que habitualmente se produce con la combinación de teclado control-\ (control-backslash), el programa escribirá un archivo ejecutable llamado `dump` en el directorio actual. Ejecutando este archivo se reanudará su programa desde el punto en que fue escrito el "dump". La siguiente sesión ilustra ésto:

```
andy@fulcrum:~/g95/g95 % cat tst.f90
  b = 0.0
  do i=1, 10
      do j=1, 3000000
          call random_number(a)
          a = 2.0*a - 1.0
          b = b + sin(sin(sin(a)))
      enddo
      print *, i, b
  enddo
  end
andy@fulcrum:~/g95/g95 % g95 tst.f90
andy@fulcrum:~/g95/g95 % a.out
 1 70.01749
 2 830.63153
```

```
 3 987.717
 4 316.48703
 5 -426.53815
 6 25.407673      (control-\ hit)
Process dumped
 7 -694.2718
 8 -425.95465
 9 -413.81763
 10 -882.66223
andy@fulcrum:~/g95/g95 % ./dump
Restarting
............Jumping
 7 -694.2718
 8 -425.95465
 9 -413.81763
 10 -882.66223
andy@fulcrum:~/g95/g95 %
```

Cualquier archivo abierto debe estar presente y en los mismos lugares que en el proceso original. Si ud. enlaza contra otros lenguajes, ésto puede no funcionar. Aunque el uso principal es permitirle conservar el estado de una ejecución de programa después de un reinicio, se abren otras posibilidades incluyendo la ejecución de un trabajo grande a través de una cola pequeña o ejecutar en otra máquina un proceso en curso. Se puede realizar el marcado del checkpoint ó punto de control automático de su programa fijando la variable de entorno `G95_CHECKPOINT` con el número de segundos a esperar entre volcados (dumps). El valor de 0 significa que no realice volcados. Los nuevos archivos de punto de control (*checkpoint*)sobreescriben los viejos.

# Compilación Eficiente

Considere el módulo `foo` cuyo código fuente resida en el archivo `foo.f95`. Podemos distinguir dos tipos de cambios en `foo.f95`:

1. Cambios que alteran el uso de `foo`, p.ej., cambiando la interfaz a un procedimiento;
2. Cambios que no alteran el uso de `foo`, sólo a su implementación, p.ej., corrigiendo un error en el código interno del procedimiento.

Ambos tipos de cambios afectarán generalmente a los contenidos del archivo objeto `foo.o`, pero sólo el primer tipo de cambio puede alterar el contenido de `foo.mod`. Cuando recompila un módulo, g95 es lo bastante inteligente para detectar cuándo el archivo `.mod` mecesita actualizarse: despúes de los cambios del tipo 2, se conserva el antiguo `.mod`.

Esta característica de g95 previene compilaciones innecesarias cuando se construye un programa grande. Efectivamente, suponga que muchos archivos de código fuente dependan de `foo.mod`, ya sea directamente (a través de la sentencia `USE FOO`) ó indirectamente (usando un módulo que use `foo`, o usando un módulo que use un módulo que use `foo`, etc). Un cambio del tipo 1 en `foo.f95` provocará una recompilación de de todos los archivos de código fuente dependentes; afortunadamente, dichos cambios son probablemente infrecuentes. Los cambios más comunes tipo 2 provocan sólo la recompilación del mismo `foo.f95`, después de lo cual el nuevo archivo objeto `foo.o` puede ser inmediatamente enlazado con los otros archivos objeto existentes para crear el programa ejecutable actualizado.

# Extensiones de las Funciones Intrínsecas de G95

ACCESS

```
INTEGER FUNCTION access(filename, mode)
    CHARACTER(LEN=*) :: filename
    CHARACTER(LEN=*) :: mode
END FUNCTION access
```

Verifica si el archivo *filename* puede ser accedido en el modo especificado, donde *mode* es una o más de las letras `rwxRWX`. Devuelve cero si los permisos con correctos, y un valor distinto de cero si algo no es correcto.

ALGAMA

```
REAL FUNCTION algama(x)
    REAL, INTENT(IN) :: x
END FUNCTION algama
```

Devuelve el logaritmo natural de $\Gamma(x)$. `ALGAMA` es una función genérica que toma cualquier tipo real.

BESJ0

```
REAL FUNCTION besj0(x)
    REAL, INTENT(IN) :: x
END FUNCTION besj0
```

Devuelve la función de Bessel de orden cero del primer tipo. Ésta función es genérica.

BESJ1

```
REAL FUNCTION besj1(x)
    REAL, INTENT(IN) :: x
END FUNCTION besj1
```

Devuelve la función de Bessel de orden uno del primer tipo. Ésta función es genérica.

BESJN

```
REAL FUNCTION besjn(n,x)
    INTEGER, INTENT(IN) :: n
    REAL, INTENT(IN) :: x
END FUNCTION besjn
```

Devuelve la función de Bessel de orden $n$ del primer tipo. Ésta función es genérica.

BESY0

```
REAL FUNCTION besy0(x)
    REAL, INTENT(IN) :: x
END FUNCTION besy0
```

Devuelve la función de Bessel de orden 0 del segundo tipo. Ésta función es genérica.

BESY1

```
REAL FUNCTION besy1(x)
    REAL, INTENT(IN) :: x
END FUNCTION besy1
```

Devuelve la función de Bessel de orden 1 del segundo tipo. Ésta función es genérica.

BESYN

```
REAL FUNCTION besyn(n,x)
    INTEGER, INTENT(IN) :: n
    REAL, INTENT(IN) :: x
END FUNCTION besyn
```

Devuelve la función de Bessel de orden $n$ del segundo tipo. Ésta función es genérica.

CHMOD

```
INTEGER FUNCTION chmod(file,mode)
    CHARACTER(LEN=*), INTENT(IN) :: file
    INTEGER, INTENT(IN) :: mode
END FUNCTION chmod
```

Cambia los permisos de unix para un archivo. Devuelve un valor distinto de cero si ocurre algún error.

DBESJ0

```
DOUBLE PRECISION FUNCTION dbesj0(x)
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION dbesj0
```

Devuelve la función de Bessel de orden 0 del primer tipo.

DBESJ1

```
DOUBLE PRECISION FUNCTION dbesj1(x)
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION dbesj1
```

Devuelve la función de Bessel de orden 1 del primer tipo.

DBESJN

```
DOUBLE PRECISION FUNCTION dbesjn(n,x)
    INTEGER, INTENT(IN) :: n
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION dbesjn
```

Devuelve la función de Bessel de orden $n$ del primer tipo.

DBESY0

```
DOUBLE PRECISION FUNCTION dbesy0(x)
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION debsy0
```

Devuelve la función de Bessel de orden 0 del segundo tipo.

DBESY1

```
DOUBLE PRECISION FUNCTION dbesy1(x)
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION dbesy1
```

Devuelve la función de Bessel de orden 1 del segundo tipo.

DBESYN

```
DOUBLE PRECISION FUNCTION dbesyn(n,x)
    INTEGER, INTENT(IN) :: n
    REAL, INTENT(IN) :: x
END FUNCTION dbesyn
```

Devuelve la función de Bessel de orden $n$ del segundo tipo.

DCMPLX

```
DOUBLE COMPLEX FUNCTION dcmplx(x,y)
END FUNCTION dcmplx
```

`CMPLX` de doble precisión, `x` y `y` pueden ser números de cualquier tipo.

DERF

```
DOUBLE PRECISION FUNCTION derf(x)
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION derf
```

Devuelve en doble precisión la función de error de `x`.

DERFC

```
DOUBLE PRECISION FUNCTION derfc(x)
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION derfc
```

Devuelve en doble precisión la función complementaria de eror de `x`.

DFLOAT

```
DOUBLE PRECISION FUNCTION dfloat(x)
END FUNCTION dfloat
```

Convierte un múmero `x` a doble precisión. Alias de la función intrínseca `DBLE`

DGAMMA

```
DOUBLE PRECISION FUNCTION dgamma(x)
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION dgamma
```

Devuelve una aproximación para $\Gamma(x)$.

DLGAMA

```
DOUBLE PRECISION FUNCTION dlgama(x)
    DOUBLE PRECISION, INTENT(IN) :: x
END FUNCTION dlgama
```

Devuelve el logaritmo natural de $\Gamma(x)$.

DREAL

```
DOUBLE PRECISION FUNCTION dreal(x)
END FUNCTION dreal
```

Convierte un número `x` a doble precisión. Alias para la `DBLE` intrínseca.

DTIME

```
REAL FUNCTION dtime(tarray)
    REAL, OPTIONAL, INTENT(OUT) :: tarray(2)
END FUNCTION dtime
```

Asigna `tarray(1)` al número de segundos transcurridos de tiempo de usuario desde que `DTIME` fue invocado la última vez. Asigna `tarray(2)` al número de segundos transcurridos de tiempo de sistema en el proceso actual desde que `DTIME` fue invocado la última vez. Devuelve la suma de los dos tiempos.

ERF

```
REAL FUNCTION erf(x)
    REAL, INTENT(IN) :: x
END FUNCTION erf
```

Devuelve la función de error de `x`. Esta función es genérica.

ERFC

```
REAL FUNCTION erfc(x)
    REAL, INTENT(IN) :: x
END FUNCTION erfc
```

Devuelve la función complementaria de error de `x`. Esta función es genérica.

ETIME

```
REAL FUNCTION etime(tarray)
    REAL, OPTIONAL, INTENT(OUT) :: tarray(2)
END FUNCTION etime
```

Asigna `tarray(1)` al número de segundos transcurridos de tiempo de usuario en el proceso actual. Asigna a `tarray(2)` el número de segundos transcurridos de tiempo de sistema en el proceso actual. Devuelve la suma de los dos tiempos.

FNUM

```
INTEGER FUNCTION fnum(unit)
    INTEGER, INTENT(IN) :: unit
END FUNCTION fnum
```

Devuelve el número descriptor de archivo correspondiente a `unit`(Unix).

FSTAT

```
INTEGER FUNCTION fstat(unit, sarray)
    INTEGER, INTENT(IN) :: unit
    INTEGER, INTENT(OUT) :: sarray(13)
END FUNCTION fstat
```

Obtiene información del archivo abierto en la unidad E/S de Fortran `unit` y la almacena en el *array* `sarray()`. Los valores de este *array* son extraidos de la estructura de estado y devueltos por fstat(2) como sigue: `sarray(1)` número de dispositivo, `sarray(2)` número de Inodo, `sarray(3)` modo de archivo, `sarray(4)` número de enlaces, `sarray(5)` uid del propietario, `sarray(6)` gid del propietario, `sarray(7)` tipo de dispositivo, `sarray(8)` tamaño de archivo, `sarray(9)` Fecha de acceso, `sarray(10)` Fecha de modificación, `sarray(11)` Fecha de cambio, `sarray(12)` Tamaño del bloque, `sarray(13)` bloques asignados.

FDATE

```
CHARACTER(LEN=*) FUNCTION fdate()
END FUNCTION fdate
```

Devuelve la fecha y hora actuales de la forma: Día Mes dd hh:mm:ss yyyy.

FTELL

```
INTEGER FUNCTION ftell(unit)
    INTEGER, INTENT(IN) :: unit
END FUNCTION ftell
```

Devuelve el desplazamiento (*offset*) actual del archivo Fortran `unit` o $-1$ si `unit` no está abierto.

GAMMA

```
REAL FUNCTION gamma(x)
    REAL, INTENT(IN) :: x
END FUNCTION gamma
```

Devuelve una aproximación para $\Gamma(x)$. `GAMMA` es una función genérica que admite cualquier tipo real.

GETCWD

```
INTEGER FUNCTION getcwd(name)
    CHARACTER(LEN=*), INTENT(OUT) :: name
END FUNCTION
```

Devuelve el directorio actual en `name`. Devuelve un valor distinto de cero si ocurre algún error.

GETGID

```
INTEGER FUNCTION getgid()
END FUNCTION getgid
```

Devuelve el gid (identificador de grupo) para el proceso actual.

GETPID

```
INTEGER FUNCTION getpid()
END FUNCTION getpid
```

Devuelve el identificador de proceso (id) para el proceso actual.

GETUID

```
INTEGER FUNCTION getuid()
END FUNCTION getuid
```

Devuelve el identificador (id) del usuario.

HOSTNM

```
INTEGER FUNCTION hostnm(name)
    CHARACTER(LEN=*), INTENT(OUT) :: name
END FUNCTION hostnm
```

Almacena en `name` el nombre del host del sistema. Devuelve un valor distinto de cero e caso de error.

IARGC

```
INTEGER FUNCTION iargc()
END FUNCTION iargc
```

Devuelve el número de argumentos de la línea de comandos (excluyendo el nombre del programa).

ISATTY

```
LOGICAL FUNCTION isatty(unit)
    INTEGER, INTENT(IN) :: unit
END FUNCTION isatty
```

Devuelve `.true.` si y solo si la unidad Fortran I/O especificada por `unit` está conectada a un dispositivo de terminal.

ISNAN

```
LOGICAL FUNCTION isnan(x)
    REAL, INTENT(IN) :: x
END FUNCTION isnan
```

Devuelve `.true.` si `x` NO es un número (NaN). Esta función es genérica.

LINK

```
INTEGER FUNCTION link(path1, path2)
    CHARACTER(LEN=*), INTENT(IN) :: path1, path2
END FUNCTION link
```

Establece un enlace duro (hard link) desde `path1` a `path2`.

LNBLNK

```
INTEGER FUNCTION lnblnk(string)
    CHARACTER(LEN=*), INTENT(IN) :: string
END FUNCTION lnblnk
```

Alias para la función standard `len_trim`. Devuelve el índice del último carácter no blanco en la cadena.

LSTAT

```
INTEGER FUNCTION LSTAT(file, sarray)
    CHARACTER(LEN=*), INTENT(IN) :: file
    INTEGER, INTENT(OUT) :: sarray(13)
END FUNCTION LSTAT
```

Si `file` es un enlace simbólico devuelve los datos del propio en lace. Vea la función `FSTAT()` para detalles adicionales. Devuelve un valor distinto de cero en caso de error.

RAND

```
REAL FUNCTION rand(x)
    INTEGER, OPTIONAL, INTENT(IN) :: x
END FUNCTION rand
```

Devuelve un número pseudo-aleatorio con distribución de probabilidad uniforme tal que $0 \leq$ `rand` $< 1$. Si `x` es 0, devuelve el siguiente número de la secuencia. Si `x` es 1, el generador vuelve a empezar llamando a `srand(0)`. Si `x` es algún otro valor, se usa como la nueva semilla con srand.

SECNDS

```
INTEGER FUNCTION secnds(t)
    REAL, INTENT(IN) :: t
END FUNCTION secnds
```

Devuelve la hora local en segundos desde la medianoche menos el valor de `t`. Esta función es genérica.

SIGNAL

```
FUNCTION signal(signal, handler)
    INTEGER, INTENT(IN) :: signal
    PROCEDURE, INTENT(IN) :: handler
END FUNCTION signal
```

Interfaz a la llamada Unix `signal`. Devuelve un valor distinto de cero en caso de error.

SIZEOF

```
INTEGER FUNCTION sizeof(object)
END FUNCTION sizeof
```

El argumento `object` es el nombre de una expresión o de un tipo. Devuelve el tamaño de `object` en bytes.

STAT

```
INTEGER FUNCTION stat(file, sarray)
    CHARACTER(LEN=*), INTENT(IN) :: file
    INTEGER, INTENT(OUT) :: sarray(13), status
END FUNCTION stat
```

Obtiene datos acerca de archivo `file` y los almacena en el *array* `sarray`. Vea la función `fstat()` para má detalles. Devuelve un valor distinto de cero en caso de error.

SYSTEM

```
INTEGER FUNCTION system(cmd)
    CHARACTER(LEN=*), INTENT(IN) :: cmd
END FUNCTION system
```

Invoca el comando externo especificado en `cmd`. Devuelve el valor del código de salida del sistema (system exit code).

TIME

```
INTEGER FUNCTION time()
END FUNCTION time
```

Devuelve la fecha/hora actual codificada como un entero en la forma de la función de UNIX `time`.

UNLINK

```
INTEGER FUNCTION unlink(file)
    CHARACTER(LEN=*), INTENT(IN) :: file
END FUNCTION unlink
```

Borra el archivo `file`. Devuelve un valor distinto de cero en caso de error.

%VAL()
Cuando es aplicada a una variable en una lista formal de argumentos, provoca que dicha variable sea pasada por valor. No se recomienda esta pseudo-función, y sólo se ha implementado por compatibilidad. El atributo de F2003 `VALUE` es el mecanismo standard para llevarlo a cabo.

%REF()
Cuando se aplica a una variable en una lista formal de argumentos, provoca que la variable sea pasada por referencia.

# Extensiones de las Subrutinas Intrínsecas de G95

ABORT

```
SUBROUTINE abort()
END SUBROUTINE abort
```

Provoca que el programa finalice con un volcado del núcleo (core dump) mandando la señal SIGABORT a sí mismo (unix).

CHDIR

```
SUBROUTINE chdir(dir)
    CHARACTER(LEN=*), INTENT(IN) :: dir
END SUBROUTINE
```

Establece el directorio actual a `dir`.

DTIME

```
SUBROUTINE dtime(tarray, result)
    REAL, OPTIONAL, INTENT(OUT) :: tarray(2), result
END SUBROUTINE dtime
```

Almacena en `tarray(1)` el número de segundos transcurridos del tiempo de usuario en el proceso actual, desde que `DTIME` fue invocado la última vez. Establece `tarray(2)` al número de segundos transcurridos del tiempo de sistema en el proceso actual, desde que `DTIME` fue invocado la última vez. Almacena en `result` la suma de los dos tiempos.

ETIME

```
SUBROUTINE etime(tarray, result)
    REAL, OPTIONAL, INTENT(OUT) :: tarray(2), result
END SUBROUTINE etime
```

Almacena en `tarray(1)` el número de segundos transcurridos del tiempo de usuario en el proceso actual. Establece `tarray(2)` al número de segundos transcurridos del tiempo de sistema en el proceso actual. Almacena en `result` la suma de los dos tiempos.

EXIT

```
SUBROUTINE exit(code)
    INTEGER, OPTIONAL, INTENT(IN) :: code
END SUBROUTINE exit
```

Sale del programa con status `code` después de cerrar las unidades E/S de Fortran. Ésta es una subrutina genérica.

FDATE

```
SUBROUTINE fdate(date)
    CHARACTER(LEN=*), INTENT(OUT) :: date
END SUBROUTINE fdate
```

Almacena en `date` la fecha y hora actual como: Día Mes dd hh:mm:ss aaaa.

FLUSH

```
SUBROUTINE flush(unit)
    INTEGER, INTENT(IN) :: unit
END SUBROUTINE flush
```

Descarga el *buffer* interno sobre el archivo fortran `unit` abierto actualmente para salida.

FSTAT

```
SUBROUTINE FSTAT(unit, sarray, status)
    INTEGER, INTENT(IN) :: unit
    INTEGER, INTENT(OUT) :: sarray(13), status
END SUBROUTINE fstat
```

Obtiene información acerca del archivo abierto en la `unit` E/S de Fortran y los almacena en el *array* `sarray()`. Almacena un valor distinto de cero en status en caso de error. Vea la función `fstat` para más información sobre cómo `sarray` is establecida.

GETARG

```
SUBROUTINE getarg(pos, value)
    INTEGER, INTENT(IN) :: pos
    CHARACTER(LEN=*), INTENT(OUT) :: value
END SUBROUTINE
```

Almacena en `value` el `pos`-ésimo argumento del intérprete de comandos.

GETENV

```
SUBROUTINE getenv(variable, value)
    CHARACTER(LEN=*), INTENT(IN) :: variable
    CHARACTER(LEN=*), INTENT(OUT) :: value
END SUBROUTINE getenv
```

Obtiene el valor de la variable de entorno `variable` y la almacena en la variable `value`.

GETLOG

```
SUBROUTINE getlog(name)
    CHARACTER(LEN=*), INTENT(OUT) :: name
END SUBROUTINE getlog
```

Devuelve el nombre de acceso para el proceso en `name`.

IDATE

```
SUBROUTINE idate(m, d, y)
    INTEGER :: m, d, y
END SUBROUTINE idate
```

Almacena en `m` el mes actual, en `d` el día del mes actual y en `y` el año actual. Esta subrutina no es muy portable a través de las implementaciones. Use la subrutina standard `DATE_AND_TIME` para el nuevo código.

LSTAT

```
SUBROUTINE lstat(file,sarray,status)
    CHARACTER(LEN=*), INTENT(IN) :: file
    INTEGER, INTENT(OUT) :: sarray(13), status
END SUBROUTINE lstat
```

Si `file` es un enlace simbólico devuelve información sobre el propio enlace. Vea `fstat()` para más detalles.

RENAME

```
SUBROUTINE rename(path1, path2, status)
    CHARACTER(LEN=*), INTENT(IN) :: path1, path2
    INTEGER, OPTIONAL, INTENT(OUT) :: status
END SUBROUTINE rename
```

Renombra el archivo `path1` a `path2`. Si se suministra el argumento `status`, se pondría a un valor distinto de cero en caso de error.

SIGNAL

```
SUBROUTINE signal(signal, handler, status)
    INTEGER, INTENT(IN) :: signal
    PROCEDURE, INTENT(IN) :: handler
    INTEGER, INTENT(OUT) :: status
END SUBROUTINE signal
```

Interfaz a la llamada de sistema de unix `signal`. Almacena en `status` un valor distinto de cero en caso de error.

SLEEP

```
SUBROUTINE sleep(seconds)
    INTEGER, INTENT(IN) :: seconds
END SUBROUTINE sleep
```

provoca que el proceso se pause por `seconds` segundos.

SRAND

```
SUBROUTINE srand(seed)
    INTEGER, INTENT(IN) :: seed
END SUBROUTINE srand
```

Reinicia el generador de números aleatorios. Vea la función `srand()` para más detalles.

STAT

```
SUBROUTINE stat(file, sarray, status)
    CHARACTER(LEN=*), INTENT(IN) :: file
    INTEGER, INTENT(OUT) :: sarray(13), status
END SUBROUTINE
```

Obtiene información acera del archivo proporcionado y la almacena en el *array* `sarray`. Vea `fstat()` para más detalles. Asigna a `status` un valor distinto de cero en caso de error.

SYSTEM

```
SUBROUTINE system(cmd, result)
    CHARACTER(LEN=*), INTENT(IN) :: cmd
    INTEGER, OPTIONAL, INTENT(OUT) :: result
END SUBROUTINE system
```

Pasa el comando `cmd` a un intérprete de comandos (shell). Si se suministra `result`, es establecido al código de salida de sistema de `cmd`.

UNLINK

```
SUBROUTINE unlink(file, status)
    CHARACTER(LEN=*), INTENT(IN) :: file
    INTEGER, INTENT(OUT) :: status
END SUBROUTINE unlink
```

Desliga (borra)el archivo `file`. En caso de error, es asignado a `status` un valor distinto de cero.