G95

It's Free Crunch Time

http://www.g95.org

G95 Merkmale:

- Freier, Fortran 95 konformer Compiler.
- Aktuelle g95 Version ist 0.91.
- GNU Open Source, GPL Lizenz.
- Ausführung der kompilierten Programme kann durch zahlreiche Umgebungsvariablen, die im Programm selbst dokumentiert sind, verändert werden.
- TR15581 - allokierbare formale Parameter, abgeleitete Typ-Komponenten.
- F2003 Prozedur-Zeiger, Struktur-Konstruktoren, Inter-Operabilität.
- F2003 Standardfunktionen und Module.
- Formale Parameter eines Unterprogramms vom Typ VALUE werden als Wert übergeben.
- Komma-Option für OPEN, READ und WRITE zur Dezimalpunkt-Auswahl.
- Eckige Klammern [und] können für Feld-Konstruktoren benutzt werden.
- IMPORT Anweisung, wird in einem Interface-Block benutzt, um Zugriff auf Grössen der übergeordneten Programmeinheit zu erhalten.
- MIN() und MAX() für Zeichenketten und numerische Typen.
- OPEN für „transparent“- oder „stream“-IO.
- Abwärtskompatibilität zu g77 „Application Binary Interface“ (ABI).
- Voreinstellung Integer 32 oder 64 Bit verfügbar.
- SYSTEM() Routine verfügbar.
- Quelltext mit Tabulatoren erlaubt.
- Symbolische Namen mit $-Option.
- Hollerith Zeichenketten.
- DOUBLE COMPLEX Erweiterung.
- Variierende Länge für benannte COMMON-Blöcke.
- gemischter Gebrauch von numerischen Typen und Zeichenketten in COMMON und EQUIVALENCE.
- INTEGER Längen: 1, 2, 4, 8.
- LOGICAL Längen: 1, 2, 4, 8.
- REAL Längen: 4, 8.
- REAL(KIND=10) für x86-kompatible Systeme. 19 Stellen Genauigkeit, Wertebereich $10^{\pm 4931}$.
- Listen-formatierte Gleitkommazahl-Ausgabe druckt minimale Zahl notwendiger Stellen, um die Zahl unterscheidbar zu halten.
- VAX Debug (D) Zeilen.
- Option für C Zeichenketten-Konstanten (z.B. 'hello\nworld').
- \ und $ Format-Beschreiber.
- VAX System-Standardfunktionen (SECNDS etc.).
- Unix System-Erweiterungsbibliothek (getenv, etime, stat, etc.).
- Erkennt nicht konforme oder nicht allokierte Felder zur Laufzeit - siehe Tabelle IV unter: `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`.
- Erkennt Speicher-Lecks - siehe Tabelle V unter: `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`.
- Rückverfolgung von Laufzeitfehlern.
- Intelligentes Compilerfeature verhindert Modul-Übersetzungskaskaden.
- Option für F-Kompatibilität. Siehe `http://www.fortran.com/F`. G95 kann als F-Compiler übersetzt werden.
- Programmunterbrechungs-Funktion für Programme unter x86/Linux.
- Obsoleter Gleitkomma-Schleifenindex ist nicht mehr erlaubt.
- Normalerweise schnelle Antwortzeiten bei Bug-Meldungen.
- Kann mit GCC 4.0.3 und 4.1.1 Release-Versionen übersetzt werden.
- Erhältlich für Linux-x86, PowerPC, 64-bit Opteron, 64-bit Itanium, 64-bit Alpha.
- Erhältlich für Windows/Cygwin, MinGW und Interix.
- Erhältlich für OSX auf Power Mac G4, x86-OSX.
- Erhältlich für FreeBSD auf x86, HP-UX 11, Sparc-Solaris, x86-Solaris, OpenBSD, NetBSD, AIX, IRIX, Tru64 UNIX auf Alpha.
- Fink-Versionen sind erhältlich.
- Binär-Pakete von 'stabilen' und aktuellen Versionen für die meisten Platformen sind erhältlich unter: `http://ftp.g95.org`.

Hin und wieder treffe ich jemanden, mit dem ich Emails über g95 ausgetauscht habe. Der häufigste Kommentar, den ich dann höre, ist, welch ausserordentliche Arbeit ich dort ganz allein schaffe. Ich lache dann jedesmal und weise darauf hin, dass ich es niemals allein gemacht habe. Die Zahl der Leute, die aktiv bei g95 geholfen haben, bewegt sich vermutlich um die Tausend oder so. Die Annahme ist, dass derjenige, der den Code schreibt, auch die ganze Arbeit macht, während tatsächlich die Menschen, die Abstürze auf ein Dutzend Zeilen herunterdestillieren, unvergleichlich wertvolle Dienste leisten, ein Beitrag, der oft übersehen wird. Etwas so Kompliziertes wie einen modernen Fortran Compiler zu schreiben ist nicht etwas, was man einfach selber macht. Ich weiss es.

Wie viele Dinge, wurde g95 aus der Frustration heraus geboren. Ich schrieb den Code meiner Doktorarbeit in Fortran 77 und benutzte g77. Fortran ist eine so wunderbare Sprache für numerische Berechnungen - es ist eine „quick and dirty“ - Sprache für Leute, die mehr an der Antwort interessiert sind als an der eigentlichen Kodierung. Mein Code beinhaltete eine Reihe ziemlich fortgeschrittener Datenstrukturen - verlinkte Listen, Octrees, dünn besetzte Matrizen, Gittergenerierung für finite Elemente, Poisson-Gleichungslöser, Multipole-Entwicklung, konjugierte Gradientenminimierung und eine Menge algorithmischer Geometrie. Da ich Fortran 77 benutzte, geriet der Code ziemlich klobig und hätte sehr von dynamischer Speicherallokierung und abgeleiteten Typen profitiert. Und meine Doktorarbeit neigte sich dem Ende zu und ich brauchte eine neue Herausforderung.

Abgesehen vom Komfort fortgeschrittener Sprachmerkmale wurde ich auch durch das Werk von Bill Kahan stark inspiriert. Das, was nach Lesen seiner Arbeiten bei mir hängenblieb, ist dass, obwohl numerische Berechnungen ziemlich trickreich sind, Wege gefunden werden können, die Fehler bis auf ein Mass zu reduzieren, das keinen mehr interessiert. Der Benutzer ist hier oft der Gnade des Bibliotheks-Authors ausgeliefert.

Obwohl der Compiler der „coole“ Teil der Arbeit ist, haben mich die Bibliotheken immer viel mehr interessiert. Die Aktionen des Compilers sind durch den Standard ziemlich genau festgelegt, es sind die Bibliotheken, wo Innovation und Experimentierfreude frei walten können. Selbst im primitiven Stadium gab es schon mehr Schnickschnack in der Bibliothek als bei anderen Herstellern. Die Programmunterbrechungs-Funktion war etwas, das ich schon seit Jahren brauchte, bevor ich tatsächlich die Möglichkeit bekam, es zu implementieren.

Es war ein Riesenspass, g95 zu schreiben, und ich freue darauf, es in den vor uns liegenden Jahrzehnten zu hegen und zu pflegen.

Andy Vaught
Mesa, Arizona
October 2006

Lizenz

G95 ist unter der GNU General Public Licence (GPL) lizensiert. Für alle rechtlichen Details siehe `http://www.gnu.org/licenses/gpl.html`.

Die Laufzeit-Bibliothek ist grösstenteils GPL und enthält eine Ausnahmeregelung von der GPL, die g95-Nutzern das Recht gibt, die g95 Bibliotheken zu Codes zu linken, die nicht unter der GPL stehen und solche gelinkten Konglomerate zu verteilen, ohne dass die resultierenden Programme damit unter der GPL stehen oder in irgendeiner Form von der GPL tangiert werden.

Installationshinweise

Unix (Linux/OSX/Solaris/Irix/etc.):

Öffnen Sie eine Konsole, gehen sie zum Verzeichnis, in dem Sie g95 installieren wollen. Um g95 herunterzuladen und zu installieren, geben Sie folgende Kommandos ein:

```
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
```

Die folgenden Dateien und Verzeichnisse sollten vorhanden sein:

```
./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
```

Das File cc1 ist ein symbolischer Verweis auf f951 im gleichen Verzeichnis.

Cygwin

Die Option -mno-cygwin ermöglicht der Cygwin-Version von g95 ausführbare Programme zu bauen, die keinen Zugriff auf cygwin1.dll benötigen, um zu arbeiten, und daher leicht auf anderen Systemen laufen können. Ausserdem sind diese ausführbaren Programme frei von Restriktionen aus dem Anhang der GPL. Um eine Cygwin-Version mit einer arbeitsfähigen -mno-cygwin Option zu installieren, benötigen Sie installierte mingw-Bibliotheken, erhältlich auf der Cygwin-Seite `http://www.cygwin.com`. Laden Sie das Binärarchiv von `http://ftp.g95.org/g95-x86-cygwin.tgz` in Ihr Cygwin-Wurzelverzeichnis (normalerweise C:\Cygwin) herunter. Starten Sie eine Cygwin-Sitzung, und führen Sie folgende Kommandos aus:

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

Dies installiert das g95 Programm in der /usr/local/bin Verzeichnisstruktur. Achtung: Benutzen Sie nicht Winzip, um die Dateien aus dem Tar-Archiv zu extrahieren, andernfalls werden die symbolischen Verweise nicht korrekt gesetzt.

MinGW

Die g95-Binärdateien für die MS-Windows-Umgebung sind selbstextrahierende Installationsprogramme. Zwei Versionen sind zur Zeit verfügbar. Windows 98 Benutzer sollten das mit der gcc Version 4.0.3 gebaute g95 Paket herunterladen, `http://ftp.g95.org/g95-MinGW.exe`. Windows NT, XP und 2000 Benutzer können entweder das gleiche Paket oder das mit der gcc Version 4.1.1 gebaute benutzen, erhältlich unter `http://ftp.g95.org/g95-MinGW-41.exe`.

Das freie MinGW/Msys System stellt die von g95 benötigten GNU GCC Dateien bereit, darin `ld.exe` (der Linker) und `as.exe` (der GNU Assembler) aus dem binutils Paket, erhältlich unter `http://www.mingw.org`.

Das Installationsskript beherrscht zwei Arten der Installation. Wenn MinGW nicht gefunden wird, installiert es g95 zusammen mit einigen essentiellen MinGW binutils Programmen und Bibliotheken in ein vom Benutzer vorgegebenes Verzeichnis. Nehmen Sie dieses Verzeichnis in Ihren `PATH` auf, und setzen Sie die Umgebungsvariable `LD_LIBRARY_PATH` entsprechend. Wenn MinGW schon installiert ist, wird empfohlen, g95 im Wurzelverzeichnis von MinGW zu installieren (normalerweise C:\mingw), um mögliche Konflikte zu vermeiden. Wenn das Installationsskript MinGW erkennt, versucht es, im MinGW Dateisystem zu installieren. Nehmen Sie das MinGW\bin Verzeichnis in Ihren `PATH` auf, und setzen Sie die Umgebungsvariable

LD_LIBRARY_PATH=*Pfad_zu_MinGW/lib*

In Windows 98 und Windows ME ist es dazu erforderlich, die Systemdatei `autoexec.bat` zu editieren, und ein Neustart ist nötig, um die Änderungen zu aktivieren.

Hinweis für Windows XP Benutzer: MinGW erlaubt derzeit ganze 8 Megabytes für den Heap. Wenn Ihre Anwendung Zugriff auf mehr Speicher benötigt, versuchen Sie es mit den Optionen `-Wl,--heap=0x01000000` zu übersetzen. Versuchen Sie grössere Hexadezimalwerte für `--heap`, bis Ihr Programm läuft.

G95 starten

G95 ermittelt anhand der Dateiendung, wie ein Programm übersetzt werden soll. Erlaubte Dateiendungen für Fortran-Quelltexte sind .f, .F, .for, .FOR, .f90, .F90, .f95, .F95, .f03 und .F03. Die Dateiendung bestimmt, ob ein Dateiinhalt als festes oder freies Format behandelt werden soll. Für Dateien mit den Endungen .f, .F, .for und .FOR wird altes f77-kompatibles, festes Format angenommen. Dateien mit den Endungen .f90, .F90, .f95, .F95, .f03 und .F03 gelten als frei formatierte Dateien. In der Voreinstellung werden Dateien mit Endungen in Grossbuchstaben vom C Preprozessor vorprozessiert, Dateien mit Endungen in Kleinbuchstaben nicht.

Die wichtigsten Optionen, um Fortran Quelltexte mit g95 zu übersetzen, sind:

`-c` Nur übersetzen, nicht linken

`-v` Zeige die tatsächlich von g95 aufgerufenen Programme und ihre Argumente. Besonders brauchbar, um Pfad-Probleme zu verfolgen.

`-o` Spezifiziert den Namen der Ausgabe-Datei, entweder die Objekt-Datei oder das ausführbare Programm. Wenn keine Ausgabe-Datei angegeben ist, heisst die Ausgabe-Datei a.out auf Unix oder a.exe auf Windows Systemen.

Einfache Beispiele:

`g95 -c hello.f90`
Übersetzt `hello.f90` in eine Objekt-Datei `hello.o`.

`g95 hello.f90` Übersetzt `hello.f90` und bindet es zu einem ausführbaren Programm namens a.out (auf Unix) oder a.exe (auf MS-Windows Systemen) zusammen.

Zusammenfassung der Optionen:

g95	`[ -c | -S | -E ]`	Übersetze und assembliere | Erzeuge Assembler Code | Liste den Quelltext
	`[ -g ] [ -pg ]`	Debug Optionen
	`[ -O[`*n*`] ]`	Optimierungsstufe, *n*=0,1,2,3
	`[ -s ]`	Debug Symbole entfernen
	`[ -W`*warn*` ] [-pedantic]`	Schalter für Warnungen
	`[ -I`*dir*` ]`	zu durchsuchendes Include-Verzeichnis
	`[ -L`*dir*` ]`	zu durchsuchendes Bibliotheks-Verzeichnis
	`[ -D macro[=wert]... ]`	Definiere Makro
	`[ -U macro ]`	lösche Makro
	`[ -f option ...]`	allgemeine Übersetzungs-Optionen
	`[ -m Maschinen-Option ]`	Maschinen-spezifische Optionen. Siehe GCC Handbuch.
	`[ -o Ausgabe-Datei ]`	Name der Ausgabe-Datei.
	`Eingabe-Datei`	

G95 Optionen

Aufruf: g95 [Optionen] Datei ...

`-pass-exit-codes`	Beende mit dem höchsten Fehlercode einer Phase.
`--help`	Zeige diese Informationen.
`--target-help`	Zeige Maschinen-spezifische Kommandozeilen-Optionen. (Verwenden Sie '-v –help', um Optionen der Subprozesse anzuzeigen).
`-dumpspecs`	Zeige alle eingebauten Spezifikations-Strings.
`-dumpversion`	Zeige Version des Compilers.
`-dumpmachine`	Zeige Ziel-Prozessor des Compilers.
`-print-search-dirs`	Zeige die Verzeichnisse im Compiler-Suchpfad.
`-print-libgcc-file-name`	Zeige den Namen der Compiler-Bibliothek.
`-print-file-name=`*lib*	Zeige den vollen Pfad zur Bibliothek *lib*.
`-print-prog-name=`*prog*	Zeige den vollen Pfad zum Compiler-Programm *prog*.
`-print-multi-directory`	Zeige das Wurzelverzeichnis für Versionen von libgcc.
`-print-multi-lib`	Zeige die Zuordnung zwischen Kommandozeilen-Option und multiplen Bibliotheks-Verzeichnis-Suchpfaden.
`-print-multi-os-directory`	Zeige den relativen Pfad zu System-Bibliotheken.
`-Wa,`*Optionen*	Übergebe die durch Komma getrennten *Optionen* zum Assembler.
`-Wp,`*Optionen*	Übergebe die durch Komma getrennten *Optionen* zum Preprozessor.
`-Wl,`*Optionen*	Übergebe die durch Komma getrennten *Optionen* zum Linker.
`-Xassembler` *arg*	Übergebe *arg* zum Assembler.
`-Xpreprocessor` *arg*	Übergebe *arg* zum Preprozessor.
`-Xlinker` *arg*	Übergebe *arg* zum Linker.
`-save-temps`	Lösche temporäre Dateien nicht.
`-pipe`	Verwende Pipes statt temporärer Dateien.
`-time`	Zeitnahme für jeden Sub-Prozess. Für einige Platformen nicht verfügbar (MinGW, OSX).
`-specs=`*Datei*	Übersteuere eingebaute Spezifikationen mit dem Inhalt von *Datei*.
`-std=`*Standard*	Verwende *Standard* für Eingabe-Quelltext.
`-B` *Verzeichnis*	Hänge *Verzeichnis* an Compiler-Suchpfade an.
`-b` *Maschine*	Rufe gcc für Zielarchitektur *Maschine* auf, falls installiert.
`-V` *Version*	Rufe gcc *Version* auf, falls installiert.
`-v`	Zeige die Programme, die von g95 aufgerufen werden.
`-M`	Erzeuge Datei-Abhängigkeiten für Makefile auf Standard-Ausgabe.
`-###`	Wie -v, aber Optionen und Kommandos nicht ausgeführt.
`-E`	Nur Preprozessierung; nicht übersetzen, assemblieren oder binden.
`-S`	Nur übersetzen; nicht assemblieren oder binden.
`-c`	Nur übersetzen und assemblieren, aber nicht binden.
`-o` *Datei*	Ausgabe in *Datei*.
`-x` *Sprache*	*Sprache* der folgenden Eingabe-Dateien spezifizieren. Erlaubte Sprachen sind: c, c++, assembler, none; 'none' heisst, die Sprache anhand der Dateiendung zu erraten.

Optionen, die mit `-g, -f, -m, -O, -W oder --param` beginnen, werden automatisch zu den jeweiligen Sub-Programmen, die g95 aufruft, durchgereicht. Um diesen Prozessen andere Optionen zu übergeben, müssen die W*letter* Optionen benutzt werden. Anweisungen zum Fehler-Report siehe: `http://www.g95.org`.

In der Voreinstellung werden g95-übersetzte Programme nicht optimiert. Das n in `-O`n spezifiziert die Optimierungsstufe zwischen 0 und 3. 0 bedeutet keine Optimierung, und höhere Zahlen steigende Optimierungsstufen. Einschalten der Optimierung erlaubt dem Compiler Code-Änderungen zur Beschleunigung. Die Ergebnisse einer Berechnung können dadurch oft in subtiler Weise beeinflusst werden. `-O` ist das Gleiche wie `-O1`.

Signifikante Beschleunigung können mit Angabe von wenigstens `-O2 -march=`*arch* erzielt werden, wobei *arch* die Zielarchitektur Ihres Prozessors ist, d.h. `pentium4, athlon, opteron etc.`. Weitere typische Fortran-Optionen sind `-funroll-loops, -fomit-frame-pointer, -malign-double` und `-msse2`. Für Informationen über alle GCC Optionen, die beim Übersetzen mit g95 zur Verfügung stehen, siehe:
`http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc`.

Preprozessor Optionen

G95 kann Dateien verarbeiten, die C Preprozessor-Anweisungen enthalten.

`-cpp`	Eingabe-Dateien durch C Preprozessor verarbeiten
`-no-cpp`	Eingabe-Dateien nicht durch C Preprozessor verarbeiten
`-D name[=Wert]`	Definiere ein Proprozessor-Makro
`-U name`	Lösche ein Preprozessor-Makro
`-E`	Zeige nur den vorprozessierten Quelltext
`-I` *Verzeichnis*	Hänge *Verzeichnis* an die Include- und Moduldatei-Suchpfade an. Verzeichnisse werden in folgender Reihenfolge nach Dateien durchsucht: Verzeichnis der Quelltextdatei, das aktuelle Verzeichnis, Verzeichnisse, die mit `-I` spezifiziert wurden, Verzeichnisse in der Umgebungsvariable `G95_INCLUDE_PATH` und zuletzt die System-Verzeichnisse.

Fortran Optionen

`-Wall`	Schalte die meisten Warnungen ein.
`-Werror`	Behandle Warnungen als Fehler.
`-Werror=`*Warnungen*	Behandle Komma-getrennte Liste der *Warnungen* als Fehler.
`-Wextra`	Schalte Warnungen ein, die nicht per `-Wall` eingeschaltet werden. Dies sind: `-Wobsolescent, -Wunused-module-vars, -Wunused-module-procs -Wunused-internal-procs, -Wunused-parameter, -Wunused-types -Wmissing-intent` und `-Wimplicit-interface`.
`-Wglobals`	Prüfe Prozedur-Aufruf und -Definition gegeneinander in der gleichen Quelldatei. Voreingestellt, setzen Sie `-Wno-globals`, um dies abzuschalten.
`-Wimplicit-none`	Das gleiche wie `-fimplicit-none`.
`-Wimplicit-interface`	Warnung, wenn implizite Schnittstellen benutzt werden.
`-Wline-truncation`	Warnung, wenn Quelltextzeilen abgeschnitten werden.
`-Wmissing-intent`	Warnung, wenn Intent für formale Parameter fehlt.
`-Wobsolescent`	Warnung, wenn obsolete Sprachelemente benutzt werden.
`-Wno=`*Warnungen*	Unterdrücke alle *Warnungen* in Komma-getrennter Liste.
`-Wuninitialized`	Warnung, wenn Variablen benutzt werden, bevor sie initialisiert sind. Benötigt `-O2`.
`-Wunused-internal-procs`	Warnung, wenn eine interne Prozedur nie benutzt wird.
`-Wunused-vars`	Warnung, wenn Variablen nicht benutzt werden.
`-Wunused-types`	Warnung, wenn Modul-Typen nicht benutzt werden. Nicht von `-Wall` eingeschaltet.
`-Wunset-vars`	Warnung, wenn Variablen nicht gesetzt werden.
`-Wunused-module-vars`	Warnung, wenn Modul-Variablen nicht benutzt werden. Sinnvoll, um `ONLY` Listen zu erstellen.
`-Wunused-module-procs`	Warnung, wenn Modul-Prozeduren nicht benutzt werden. Sinnvoll, um `ONLY` Listen zu erstellen.
`-Wunused-parameter`	Warnung, wenn Konstanten nicht benutzt werden. Nicht von `-Wall` eingeschaltet.
`-Wprecision-loss`	Warnung, wenn Genauigkeitsverluste in impliziten Typumwandlungen auftreten.
`-fbackslash`	Backslash ('\') als Escape-Steuerzeichen interpretieren. Voreingestellt, benutzen Sie `-fno-backslash`, um Backslash als normales Zeichen zu verwenden.
`-fc-binding`	Ausgabe der C Prozedur-Prototypen auf Standardausgabe.
`-fd-comment`	D Zeilen als ausführbare Anweisung im festen Format behandeln.
`-fdollar-ok`	Dollar-Zeichen als Bestandteil von Namen erlauben.
`-fendian=`*Wert*	Byteanordnung für unformatiertes Lesen und Schreiben vorgeben. Der *Wert* muss `big` oder `little` sein. Übersteuert Umgebungsvariablen zur Laufzeit.
`-ffixed-form`	Behandle Quelltextdatei im festen Format.
`-ffixed-line-length-132`	132 Zeichen Zeilenbreite im festen Format.
`-ffixed-line-length-80`	80 Zeichen Zeilenbreite im festen Format.
`-ffree-form`	Behandle Quelltextdatei im freien Format
`-ffree-line-length-huge`	Erlaube sehr lange Quelltextzeilen (10k).
`-fimplicit-none`	Verbiete implizite Typ-Vergabe, ausser durch explizite `IMPLICIT`-Anweisungen.
`-fintrinsic-extensions`	Erlaube g95-spezifische Standardprozeduren auch im `-std=` Modus.
`-fintrinsic-extensions=`	Erlaube ausgewählte spezifische Standardprozeduren auch im `-std=` Modus. In der Komma-separierten Liste spielt Gross-Kleinschreibung keine Rolle.

`-fmod=`*Verzeichnis* Schreibe Modul-Dateien in `Verzeichnis`.
`-fmodule-private` Sichtbarkeit von Modul-Grössen auf `PRIVATE` setzen.
`-fmultiple-save` Erlaube mehrfache Angabe des `SAVE` Attributs.
`-fone-error` Stoppe Übersetzung nach dem ersten Fehler.
`-ftr15581` Erlaube `TR15581`-Erweiterung zu allokierbaren Feldern auch im `-std=F` oder `-std=f95` Modus.
`-std=F` Warnungen, wenn nicht F-konforme Features benutzt werden.
`-std=f2003` Strenge Fortran 2003 Prüfung.
`-std=f95` Strenge Fortran 95 Prüfung.
`-i4` Setze Länge nicht explizit spezifizierter Integer auf kind=4 (32 Bit).
`-i8` Setze Länge nicht explizit spezifizierter Integer auf kind=8 (64 Bit).
`-r8` Setze Länge nicht explizit spezifizierter Gleitkommazahlen auf doppelte Genauigkeit.
`-d8` Setzt `-i8` und `-r8`.

Optionen zur Code-Generierung

`-fbounds-check` Feld- und Zeichenketten-Grenzen zur Laufzeit prüfen.
`-fcase-upper` Alle öffentlichen Symbole in Grossbuchstaben wandeln.
`-fleading-underscore` Setze einen Unterstrich vor alle öffentlichen Namen.
`-fonetrip` Alle `DO`-Schleifen wenigstens einmal durchlaufen. (Fehlerhaftes `FORTRAN 66`.)
`-fpack-derived` Versuche, abgeleitete Typen so kompakt wie möglich anzulegen. Benötigt weniger Speicher, kann aber langsamer sein.
`-fqkind=`*n* Setze die Länge einer Gleitkommazahl mit 'q'-Exponent auf *n*.
`-fsecond-underscore` Hänge einen zusätzlichen Unterstrich an Namen mit Unterstrich (voreingestellt). Benutzen Sie `-fno-second-underscore`, um dies zu unterdrücken.
`-fshort-circuit` Zweiten Operanden in `.AND.` und `.OR.` nicht berechnen, wenn Wert des Ausdrucks bereits aus dem ersten Operanden folgt.
`-fsloppy-char` Fehler unterdrücken, wenn Zeichenketten-Typen mit numerischen Daten beschrieben werden; Erlaube Vergleich von `INTEGER` und `CHARACTER` Variablen.
`-fstatic` Lokale Variablen statisch anlegen, wenn möglich. Dies ist nicht das Gleiche wie statisches Binden (`-static`).
`-ftrace=` `-ftrace=frame` fügt Code ein, um Rückverfolgung bei Abstürzen zu ermöglichen. Dies wird Ihr Programm verlangsamen. `-ftrace=full` erlaubt es zusätzlich, die Zeilennummer arithmetischer Ausnahmefehler zu finden (langsamer). Voreingestellt ist `ftrace=none`.
`-funderscoring` Unterstrich an globale Namen anhängen. Voreingestellt, benutzen Sie `-fno-underscoring`, um dies abzuschalten.
`-max-frame-size=`*n* Grösse eines maximalen Stack-Frames in Bytes, bevor Felder dynamisch angelegt werden.
`-finteger=`*n* Setze nicht-initialisierte skalare Integer Variablen auf *n*.
`-flogical=`*Wert* Setze nicht-initialisierte skalare logische Variablen. Erlaubte *Wert*e sind `none, true` und `false`.
`-freal=`*Wert* Setze nicht-initialisierte skalare Gleitkomma- und komplexe Variablen. Erlaubte *Wert*e sind `none, zero, nan, inf, +inf` und `-inf`.
`-fpointer=`*Wert* Setze nicht-initialisierte skalare Zeiger. Erlaubte *Wert*e sind `none, null` und `invalid`.
`-fround=`*Wert* Steuert Rundung bei der Übersetzung. *Wert* darf `nearest, plus, minus` und `zero` sein. Voreinstellung ist Rundung auf `nearest`, `plus` ist Rundung Richtung +Unendlich, `minus` ist Rundung Richtung -Unendlich, `zero` ist Rundung Richtung Null.
`-fzero` Initialisiere numerische Variablen auf Null, logische Variablen auf false und Zeiger auf `NULL()`. Die anderen Initialisierungs-Optionen überschreiben diese.

Verzeichnis-Optionen

`-I` *Verzeichnis* Hänge *Verzeichnis* an die Include- und Moduldatei-Suchpfade an.
`-L`*Verzeichnis* Hänge *Verzeichnis* an die Bibliotheks-Suchpfade an.
`-fmod=`*Verzeichnis* Schreibe Modul-Dateien in `Verzeichnis`.

Umgebungsvariablen

Die g95 Laufzeit-Umgebung bietet viele Optionen, um das Verhalten Ihres Programms zu optimieren, wenn es denn einmal läuft. Dies ist über Umgebungsvariablen steuerbar. Wenn Sie ein g95-kompiliertes Programm mit der Option `--g95` laufen lassen, werden diese Optionen auf der Standardausgabe gelistet. Die Werte der verschiedenen Variablen sind immer Zeichenketten, aber diese Zeichenketten werden als Integer oder Boolsche Werte interpretiert. Nur das erste Zeichen eines Boolschen Wertes wird geprüft und muss `'t', 'f', 'y', 'n', '1'` oder `'0'` sein (Grossbuchstaben werden auch akzeptiert). Ist ein Wert unbrauchbar, wird kein Fehler gemeldet, und die Voreinstellung wird benutzt. Für GCC-Umgebungsvariablen, die von g95 verwendet werden, wie etwa `LIBRARY_PATH`, nehmen Sie bitte die GCC-Dokumentation zu Hilfe.

`G95_STDIN_UNIT`	Integer	Vorverbundene Dateinummer für Standard-Eingabe. Keine Vorgabe, wenn negativ. Voreinstellung ist 5.
`G95_STDOUT_UNIT`	Integer	Vorverbundene Dateinummer für Standard-Ausgabe. Keine Vorgabe, wenn negativ. Voreinstellung ist 6.
`G95_STDERR_UNIT`	Integer	Vorverbundene Dateinummer für Standard-Fehlerausgabe. Keine Vorgabe, wenn negativ. Voreinstellung ist 0.
`G95_USE_STDERR`	Logisch	Bibliotheks-Ausgaben nach Standard-Fehlerausgabe statt Standard-Ausgabe. Voreinstellung `Yes`.
`G95_ENDIAN`	String	Zu verwendende Byteanordnung bei unformatiertem I/O. Erlaubte Werte sind `BIG, LITTLE` oder `NATIVE`. Voreinstellung ist `NATIVE`.
`G95_CR`	Logisch	Erzeugt Zeilenumbruch (CR) für formatierte, sequentielle Dateieinträge. Voreinstellung `TRUE` für alle Platformen ausser Cygwin/Windows, sonst `FALSE`.
`G95_INPUT_CR`	Logisch	Behandelt 'CR-LF' anstatt nur 'linefeed' (LF) als Dateieintragsende. Voreinstellung `TRUE`.
`G95_IGNORE_ENDFILE`	Logisch	Ignoriert Versuche, im sequentiellen Modus über den `ENDFILE`-Eintrag hinaus zu lesen. Voreinstellung `FALSE`.
`G95_TMPDIR`	String	Verzeichnis für temporäre Dateien. Übersteuert die `TMP` Umgebungsvariable. Wenn `TMP` nicht gesetzt ist, wird `/var/tmp` benutzt. Keine Voreinstellung.
`G95_UNBUFFERED_ALL`	Logisch	Falls `TRUE`, wird Ausgabe nicht gepuffert. Dies verzögert grosse Schreiboperationen, ist aber sinnvoll, um die sofortige Anzeige der Ausgabe zu erzwingen.
`G95_SHOW_LOCUS`	Logisch	Falls `TRUE`, drucke Dateinamen und Zeilennummer von Laufzeitfehlern. Voreinstellung ist `TRUE`.
`G95_STOP_CODE`	Logisch	Falls `TRUE`, werden stop-Codes als System-Exit Codes verwendet. Voreinstellung `TRUE`.
`G95_OPTIONAL_PLUS`	Logisch	Drucke optionale '+' bei Zahlenausgabe, wenn erlaubt. Voreinstellung `FALSE`.
`G95_DEFAULT_RECL`	Integer	Vorgegebene maximale Dateieintrags-Länge für sequentielle Dateien. Besonders sinnvoll, um Zeilenlängen für vorverbundene Dateien einzustellen. Voreinstellung ist 50 000 000.
`G95_LIST_SEPARATOR`	String	Trenner, der für listen-gesteuerte Ausgabe benutzt wird. Kann beliebige Zahl Leerzeichen und höchstens ein Komma beinhalten. Voreinstellung ist ein Leerzeichen.
`G95_LIST_EXP`	Integer	Grösster Exponent von 10, der noch kein Exponential-Format bei listengesteuerter Ausgabe benutzt. Voreinstellung 6.
`G95_COMMA`	Logisch	Benutze Komma als voreingestelltes Dezimalpunkt-Zeichen für I/O. Voreinstellung `FALSE`.
`G95_EXPAND_UNPRINTABLE`	Logisch	Normalerweise nicht-druckbare Zeichen in formatierter Ausgabe mit \-Sequenz ausgeben. Voreinstellung `FALSE`.
`G95_QUIET`	Logisch	Piepton (\a) in formatierter Ausgabe unterdrücken. Voreinstellung `FALSE`.
`G95_SYSTEM_CLOCK`	Integer	Zahl der Ticks pro Sekunde, die von der `SYSTEM_CLOCK` Standardfunktion geliefert wird. Null schaltet die Uhr aus. Voreinstellung 100 000.
`G95_SEED_RNG`	Logisch	Falls `TRUE`, wird der Zufallszahlengenerator mit einem neuen Wert initialisiert, wenn das Programm gestartet wird.
`G95_MINUS_ZERO`	Logisch	Falls `TRUE`, werden Nullen in formatierter (nicht listengesteuerten) Ausgabe ohne Minus-Zeichen gedruckt, selbst wenn der interne Wert negativ oder minus Null ist. Dies ist die herkömmliche, aber nicht standard-konforme Art des Ausdrucks von Nullen. Voreinstellung: `FALSE`.

G95_ABORT	Logisch	Falls TRUE, wird ein Speicherabbild (corefile) bei Programmabsturz angelegt. Hilfreich, um ein Problem zu lokalisieren. Voreinstellung FALSE.
G95_MEM_INIT	String	Speicher-Initialisierung. Voreingestellt ist keine Initialisierung (NONE, schneller), möglich sind NAN („Not-a-Number“) mit der Mantisse 0x00f95 oder ein anderer selbstgewählter Hexadezimalwert.
G95_MEM_SEGMENTS	Integer	Grösste Zahl der noch angelegten Speicher-Segmente, die nach Ende des Programms angezeigt werden. 0 unterdrückt diese Anzeige, kleiner 0 zeigt alle. Voreinstellung 25.
G95_MEM_MAXALLOC	Logisch	Falls TRUE, zeige die maximale Grösse angelegter Benutzer-Speichersegmente während des Laufs in Bytes. Voreinstellung FALSE.
G95_MEM_MXFAST	Integer	Maximale Anforderungsgrösse für Fastbins. Fastbins sind schneller, fragmentieren aber leichter. Voreinstellung 64 Bytes.
G95_MEM_TRIM_THRESHOLD	Integer	Grösster Speicheranteil, der gehalten werden soll, bevor er dem Betriebssystem zurückgegeben wird. -1 verhindert die Rückgabe an das System. Sinnvoll in lang laufenden Programmen. Voreinstellung 262144.
G95_MEM_TOP_PAD	Integer	Zusätzlicher Speicher, der angelegt wird, wenn Speicher vom Betriebssystem angefordert wird. Kann weitere Anforderungen beschleunigen. Voreinstellung 0.
G95_SIGHUP	String	Programm führt bei SIGHUP IGNORE, ABORT, DUMP oder DUMP-QUIT aus. Voreinstellung ABORT. Nur für Unix.
G95_SIGINT	String	Programm führt bei SIGINT IGNORE, ABORT, DUMP oder DUMP-QUIT aus. Voreinstellung ABORT. Nur für Unix.
G95_SIGQUIT	String	Programm führt bei SIGQUIT IGNORE, ABORT, DUMP oder DUMP-QUIT aus. Voreinstellung ABORT. Nur für Unix.
G95_CHECKPOINT	Integer	Die Zahl der Sekunden zwischen Programmhaltepunkt-Speicherabzügen auf x86 Linux, 0 bedeutet keine Speicherabzüge.
G95_CHECKPOINT_MSG	Logisch	Falls TRUE, drucke Meldung auf Standard-Fehlerausgabe, wenn Programm angehalten wird. Voreinstellung TRUE.
G95_FPU_ROUND	String	Setze Gleitkomma-Rundungsmodus. Zugelassene Werte sind NEAREST, UP, DOWN, ZERO. Voreinstellung ist NEAREST.
G95_FPU_PRECISION	String	Genauigkeit von Zwischenresultaten. Wert kann 24,53 oder 64 sein. Voreinstellung 64. Nur für x86 und Kompatible.
G95_FPU_DENORMAL	Logisch	Gleitkomma-Ausnahmefehler bei denormalisieren Zahlen. Voreinstellung FALSE.
G95_FPU_INVALID	Logisch	Gleitkomma-Ausnahmefehler bei ungültigen Operationen. Voreinstellung FALSE.
G95_FPU_ZERODIV	Logisch	Gleitkomma-Ausnahmefehler bei Division durch Null. Voreinstellung FALSE.
G95_FPU_OVERFLOW	Logisch	Gleitkomma-Ausnahmefehler bei Überlauf. Voreinstellung FALSE.
G95_FPU_UNDERFLOW	Logisch	Gleitkomma-Ausnahmefehler bei Unterlauf. Voreinstellung FALSE.
G95_FPU_INEXACT	Logisch	Gleitkomma-Ausnahmefehler bei Genauigkeitsverlust. Voreinstellung FALSE.
G95_FPU_EXCEPTIONS	Logisch	Falls TRUE, werden maskierte arithmetische Ausnahmefehler am Programende gelistet. Voreinstellung FALSE.
G95_UNIT_x	String	Übersteuert den voreingestellten Dateinamen für Dateinummer x. Voreingestellt ist fort.x.
G95_UNBUFFERED_x	Logisch	Falls TRUE, wird Dateinummer x nicht gepuffert. Voreinstellung FALSE.

Laufzeit-Fehlercodes

Wenn ein g95-kompiliertes Programm mit der Option --g95 gestartet wird, listet es folgende Fehlercodes auf der Standardausgabe:

-2 Ende des Dateieintrags
-1 Ende der Datei
0 Erfolgreicher Abschluss
Betriebssystem errno Codes (1-199)
200 Inkompatible Aufruf-Optionen
201 Ungültige Aufruf-Option
202 Fehlende Aufruf-Option
203 Datei schon mit anderer Dateinummer geöffnet
204 Nicht verbundene Dateinummer
205 FORMAT-Fehler
206 Ungültige ACTION angegeben
207 Lesen nach dem ENDFILE-Eintrag

208 Ungültiger Wert während des Lesens
209 Numerischer Überlauf beim Lesen
210 Zu wenig Speicher
211 Feld schon allokiert
212 Deallokation eines ungültigen Zeigers
214 Defekter Eintrag in unformatierter Datei mit sequentiellem Zugriff
215 Liest mehr Daten als Grösse des Dateieintrags (RECL)
216 Schreibt mehr Daten als Grösse des Dateieintrags (RECL)

Fortran 2003 Sprachelemente

G95 implementiert eine Reihe von Sprachelementen von Fortran 2003. Für eine Erläuterung all der neuen Elemente von Fortran 2003, siehe:
http://www.kcl.ac.uk/kis/support/cit/fortran/john_reid_new_2003.pdf.

- Die folgenden Standardprozeduren sind verfügbar: COMMAND_ARGUMENT_COUNT(), GET_COMMAND_ARGUMENT(),GET_COMMAND() und GET_ENVIRONMENT_VARIABLE.
- Gleitkommazahl-Schleifenindex in einfacher und doppelter Genauigkeit sind in g95 nicht implementiert.
- Eckige Klammern [und] können alternativ zu (/ und /) für Feld-Konstruktoren genutzt werden.
- TR-15581 - allokierbare Felder abgeleiteten Typs. Erlaubt den Gebrauch des ALLOCATABLE Attributs für formale Argumente, Resultate von Funktionen und Struktur-Komponenten.
- Stream I/O - F2003 Stream-Zugriff ermöglicht einem Fortran-Programm, binäre Dateien zu lesen, ohne sich über die Struktur von Dateieinträgen Gedanken machen zu müssen. Clive Page hat etwas Dokumentation zu diesem Feature geschrieben, verfügbar unter:
  http://www.star.le.ac.uk/ cgp/streamIO.html.
- IMPORT Anweisung. Wird in einem Interface-Block gebraucht, um Zugriff auf Grössen der übergeordneten Programmeinheit zu erhalten.
- Europäische Regel für Gleitkommazahlen - DECIMAL='KOMMA' Parameter für OPEN, READ und WRITE Anweisungen erlaubt Ersatz des Dezimalpunkts in Gleitkommazahlen durch ein Komma.
- MIN() und MAX() arbeitet sowohl mit Zeichen als auch mit numerischen Typen.
- Das VALUE Attribut für ein formales Argument eines Unterprogramms sorgt dafür, dass das aktuelle Argument per Wert übergeben wird.
- F2003 Struktur-Konstruktoren werden unterstützt.
- F2003 Prozedur-Zeiger werden unterstützt.
- F2003 BIND(C) Anweisung, ISO_C_BINDING Modul zur erleichterten Interoperabilität mit C.

Schnittstellen mit G95 Programmen

Gleichwohl g95 selbstständige ausführbare Programme produziert, ist es manchmal wünschenswert, an andere Programme anzukoppeln, typischerweise C. Die erste Schwierigkeit, die einem „mehrsprachigem“ Programm begegnet, sind die Namen der öffentlichen Symbole. G95 folgt hier der f2c Übereinkunft, einen Unterstrich an öffentliche Namen anzuhängen, oder zwei Unterstriche, wenn der Namen selbst schon einen Unterstrich enthält. Die Optionen -fno-second-underscore und -fno-underscoring können hilfreich dabei sein, g95 zu veranlassen, Namen zu generieren, die mit Ihrem C Compiler kompatibel sind. Benutzen Sie das nm Programm, um in die von beiden Compilern erzeugten .o Dateien hineinzuschauen. G95 überführt öffentliche Namen in Kleinbuchstaben, sofern nicht die Option -fupper-case angegeben wird, was dann dazu führt, dass alles in Grossbuchstaben erscheint. Namen in Modulen werden in *Modulname*_MP_*Name_der_Grösse* überführt.

Nach dem Binden gibt es zwei Fälle: Fortran ruft C-Unterprogramme, oder C ruft Fortran-Unterprogramme auf. Im Falle C ruft Fortran-Unterprogramme rufen die Fortran-Routinen oft Fortran-Bibliotheks-Routinen auf, die erwarten, das der Heap in irgendeiner Weise initialisiert ist. Um die manuelle Initialisierung von C aus zu veranlassen, rufen Sie g95_runtime_start() auf, um die Fortran-Bibliothek zu initialisieren, und g95_runtime_stop(), wenn Sie fertig sind. Der Prototyp von g95_runtime_start() ist:

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

Die Bibliothek muss in der Lage sein, Kommandozeilen-Optionen zu verarbeiten. Wenn Ihnen dies zu schwierig ist oder Ihr Programm keine Kommandozeilen-Argumente benötigt, übergeben Sie einfach `argc=0` und `argv=NULL`. Unter OSX müssen Sie `-lSystemStubs` einfügen, wenn Sie g95 benutzen, um den Linker laufen zu lassen und C-Objektdateien zu binden.

F2003 stellt eine Reihe von Features zur Verfügung, die die Kopplung mit C erleichtern. Das `BIND(C)` Attribut erlaubt die Erzeugung von Fortran-Symbolen, die leichter von C (oder anderen Sprachen) referenziert werden können. Zum Beispiel:

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

Diese Form erzeugt ein Symbol namens `foo` ohne jedwede Unterstrich-Verstümmelung. Alle Zeichen sind in Kleinbuchstaben. Eine ähnliche Form ist:

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

Dies bewirkt, dass der Name des Symbols jetzt `Foo1` ist. Innerhalb Fortran wird bdie Routine nach wie vor mit dem Namen `foo`, `FOO` oder jeder anderen Gross/Kleinschreibung angesprochen.

C-Programme übergeben Argumente per Wert, während Fortran-Programme dies per Referenz bewerkstelligen. F2003 stellt dazu das `VALUE` Attribut zur Verfügung, mit dem man festlegen kann, das ein formales Argument per Wert übergeben wird. Ein Beispiel wäre:

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

Eine so definierte Routine kann nach wie vor von Fortran aus aufgerufen werden, allerdings mit der Einschränkung, das formale Argumente nicht mehr mit aktuellen Argumenten verknüpft sind, Veränderung eines formalen Arguments verändert dann nicht mehr das aktuelle Argument.

Auf globale Variablen kann auf ähnliche Weise zugegriffen werden. Das folgende Unterprogramm druckt den Wert der `VAR` Variable, auf die anders von Fortran aus nicht zugegriffen werden könnte:

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

Wo Fortran annimmt, das es gleiche Typen verschiedener Arten (Längen) geben kann, definiert C alles als unterschiedliche Typen. Damit das gleiche Objekt spezifiziert werden kann, bietet F2003 das Standardmodul `ISO_C_BINDING`, das Abbildungen zwischen Fortran Arten und C Typen enthält. Wenn dieses per `USE` eingebunden wird, sind die folgenden Konstanten definiert:

`c_int`	Integer Länge für C `int`
`c_short`	Integer Länge für C `short`
`c_long`	Integer Länge für C `long`
`c_long_long`	Integer Länge für C `long long`
`c_signed_char`	Integer Länge für C `char`
`c_size_t`	Integer Länge für `size_t`
`c_intptr_t`	Integer Länge der gleichen Grösse wie C Zeiger
`c_float`	Gleitkommazahl Länge für C `float`
`c_double`	Gleitkommazahl Länge für C `double`

Es gibt noch viele andere Dinge in ISO_C_BINDING. Mit Hilfe dieses Moduls kann man dann so ein Programm schreiben:

```
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
   ...
```

Aufruf des Zufallszahlengenerators

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

Gibt einen REAL Skalar oder ein Feld von REAL Zufallszahlen in harvest zurück, 0$\leq$ harvest $\leq$ 1.
Initialisierung des Zufallszahlengenerators:

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

sz ist die kleinste Anzahl von Standard-Ganzzahlen, die benötigt werden, um die Startwerte zu halten. g95 gibt vier zurück. Argument pt ist ein Feld von Standard-Ganzzahlen der Grösse n1$\geq$sz, das die vom Benutzer vorgegebenen Startwerte enthält. Argument gt ist ein Feld von Standard-Ganzzahlen der Grösse n2$\geq$sz, das die momentanen Startwerte enthält.

Wenn RANDOM_SEED() ohne Argumente aufgerufen wird, werden die Startwerte auf Basis der Systemzeit initialisiert. Dies kann man dazu benutzen, Zufallszahl-Sequenzen zu erzeugen, die bei jedem Aufruf des Programms verschieden sind. Der Startwert wird auch dann auf Systemzeit-Basis initialisiert, wenn die G95_SEED_RNG Umgebungsvariable auf TRUE gesetzt ist. Wenn keine dieser Bedingungen erfüllt ist, wird RANDOM_NUMBER() immer die gleiche Sequenz erzeugen.

Der zugrundeliegende Generator ist der xor-shift Generator von George Marsaglia.

Vordefinierte Preprozessor-Makros

Die folgenden Makros sind immer definiert:

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

Die bedingten Makros sind:

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

Programmunterbrechungs-Funktion

Auf x86 Linux-Systemen kann ein g95-kompiliertes Programm angehalten und wiederaufgesetzt werden. Wenn Sie ein Programm mit einem `QUIT`-Signal, das normalerweise an Control-Backslash gebunden ist, unterbrechen, schreibt das Programm eine ausführbare Datei namens `dump` in das aktuelle Verzeichnis. Wenn Sie dieses Programm starten, wird Ihr ursprüngliches Programm von der Stelle an weiterrechnen, an der die `dump`-Datei geschrieben wurde. Die folgende Sitzung illustriert dieses Verhalten:

```
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 %
```

Sämtliche offenen Dateien müssen vorhanden und an derselben Stelle zu finden sein wie im Originalprozess. Wenn Sie zu anderen Sprachen binden, wird dies möglicherweise nicht funktionieren. Obgleich die hauptsächliche Anwendung sein wird, den Zustand eines Prozess über einen Neustart hinaus zu erhalten, gibt es andere Möglichkeiten wie z.B.

einen Langläufer durch eine zeitbegrenzte Queue zu bekommen oder einen laufenden Prozess auf eine andere Maschine zu transferieren. Automatisches Checkpointing Ihres Programs erreichen Sie durch Belegen der Umgebungsvariable `G95_CHECKPOINT` mit der Anzahl der Sekunden, die zwischen zwei Speicherabzügen verstreichen soll. Der Wert 0 unterdrückt Speicherabzüge. Neue Checkpoint-Dateien überschreiben alte Checkpoint-Dateien.

Intelligentes Übersetzen

Nehmen wir ein Modul, dessen Quellcode in der Datei `foo.f95` steht. Wir können zwischen zwei Typen der Änderung unterscheiden:

1. Änderungen, die den Gebrauch des Moduls verändern, z.B. die Veränderung der Schnittstelle einer Prozedur;
2. Änderungen, die den Gebrauch des Moduls nicht verändern, sondern nur seine Implementierung, z.B. das Beheben eines Fehlers in einem Programmblock.

Beide Typen der Änderung werden im allgemeinen den Inhalt der Objektdatei `foo.o` betreffen, aber nur der erste Typ kann den Inhalt von `foo.mod` verändern. Wenn ein Modul erneut übersetzt wird, ist g95 intelligent genug, um zu entscheiden, ob die `.mod` Datei erneuert werden muss oder nicht: Nach Änderungen des Typs 2 wird die alte `.mod` Datei beibehalten.

Diese Eigenschaft von g95 beugt unnötigen Übersetzungskaskaden vor, wenn ein grosses Programm zusammengebaut wird. Angenommen, viele verschiedene Quellcode-Dateien hängen von `foo.mod` ab, entweder direkt (wegen einer `USE FOO` Anweisung) oder indirekt (durch den Gebrauch eines Moduls, das seinerseits `foo` benutzt, oder durch Gebrauch eines Moduls, das ein Modul benutzt, das wiederum von `foo` abhängt, etc.). Eine Änderung vom Typ 1 an `foo.f95` wird eine Neuübersetzung aller abhängigen Quellcodedateien in Gang setzen; glücklicherweise sind solche Änderungen selten. Die weitaus häufigeren Änderungen vom Typ 2 werden nur eine Neuübersetzung von `foo.f95` veranlassen, und danach kann die Objektdatei `foo.o` sofort mit den anderen schon existierenden Objektdateien zusammengebunden werden, um das ausführbare Programm auf den neuesten Stand zu bringen.

G95 Erweiterungen zu Standardfunktionen

ACCESS

```
INTEGER FUNCTION access(Name, Modus)
    CHARACTER(LEN=*) :: Name
    CHARACTER(LEN=*) :: Modus
END FUNCTION ACCESS
```

Prüft, ob auf die Datei `Name` mit dem angegeben `Modus` zugegriffen werden kann, wobei `Modus` einer oder mehrere der Buchstaben `rwxRWX` sein muss. Gibt 0 zurück, wenn die Rechte OK sind, ungleich 0, wenn irgendetwas nicht stimmt.

ALGAMMA

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

Gibt den natürlichen Logarithmus von $\Gamma(x)$ zurück. `ALGAMMA` ist eine generische Funktion, die jeden Art von Gleitkommazahl verarbeitet.

BESJ0

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

Rückgabewert ist die Besselfunktion erster Art nullter Ordnung. Dies ist eine generische Funktion.

BESJ1

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

Rückgabewert ist die Besselfunktion erster Art erster Ordnung. Dies ist eine generische Funktion.

BESJN

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

Rückgabewert ist die Besselfunktion erster Art $n$ter-Ordnung. Dies ist eine generische Funktion.

BESY0

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

Rückgabewert ist die Besselfunktion zweiter Art nullter Ordnung. Dies ist eine generische Funktion.

BESY1

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

Rückgabewert ist die Besselfunktion zweiter Art erster Ordnung. Dies ist eine generische Funktion.

BESYN

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

Rückgabewert ist die Besselfunktion zweiter Art $n$ter Ordnung. Dies ist eine generische Funktion.

CHMOD

```
INTEGER FUNCTION chmod(Datei,Modus)
    CHARACTER(LEN=*), INTENT(IN) :: Datei
    INTEGER         , INTENT(IN) :: Modus
END FUNCTION chmod
```

Ändert Unix Zugriffsrechte für `Datei` auf `Modus`. Gibt ungleich 0 zurück, wenn ein Fehler auftritt.

DBESJ0

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

Rückgabewert ist die Besselfunktion erster Art nullter Ordnung. Dies ist eine generische Funktion.

DBESJ1

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

Rückgabewert ist die Besselfunktion erster Art erster Ordnung. Dies ist eine generische Funktion.

DBESJN

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

Rückgabewert ist die Besselfunktion erster Art $n$ter-Ordnung. Dies ist eine generische Funktion.

DBESY0

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

Rückgabewert ist die Besselfunktion zweiter Art nullter Ordnung. Dies ist eine generische Funktion.

DBESY1

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

Rückgabewert ist die Besselfunktion zweiter Art erster Ordnung. Dies ist eine generische Funktion.

DBESYN

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

Rückgabewert ist die Besselfunktion zweiter Art $n$ter Ordnung. Dies ist eine generische Funktion.

DCMPLX

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

Doppeltgenaues `CMPLX`, `x` und `y` können von beliebigen numerischen Typ jedweder Länge sein.

DERF

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

Rückgabewert ist die doppeltgenaue Fehlerfunktion von x.

DERFC

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

Rückgabewert ist die doppeltgenaue, komplementäre Fehlerfunktion von x.

DFLOAT

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

Wandelt numerisches `x` in doppelte Genauigkeit um. Anderer Name für die `DBLE` Standardfunktion.

DGAMMA

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

Gibt eine Näherung für $\Gamma(x)$ zurück.

DLGAMA

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

Rückgabewert ist der natürliche Logarithmus von $\Gamma(x)$.

DREAL

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

Wandelt numerisches `x` in doppelte Genauigkeit um. Anderer Name für die `DBLE` Standardfunktion.

DTIME

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

Setzt `TFeld(1)` gleich der Anzahl verstrichener Sekunden der Benutzer-Zeit des laufenden Prozesses seit dem letzten Aufruf von `DTIME`. Setzt `TFeld(2)` gleich der Anzahl verstrichener Sekunden der System-Zeit des laufenden Prozesses seit dem letzten Aufruf von `DTIME`. Rückgabewert ist die Summe beider Zeiten.

ERF

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

Rückgabewert ist die Fehlerfunktion von x. Dies ist eine generische Funktion.

ERFC

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

Rückgabewert ist die komplementäre Fehlerfunktion von x. Dies ist eine generische Funktion.

ETIME

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

Setzt `TFeld(1)` gleich der Anzahl verstrichener Sekunden der Benutzer-Zeit des laufenden Prozesses. Setzt `TFeld(2)` gleich der Anzahl verstrichener Sekunden der System-Zeit des laufenden Prozesses. Rückgabewert ist die Summe beider Zeiten.

FNUM

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

Gibt die Dateibeschreiber-Nummer der Dateinummer `Nummer` zurück. Rückgabewert ist -1, falls die Dateinummer nicht verbunden ist.

FSTAT

```
INTEGER FUNCTION fstat(Nummer, SFeld)
    INTEGER, INTENT(IN)  :: Nummer
    INTEGER, INTENT(OUT) :: SFeld(13)
END FUNCTION fstat
```

Ermittelt Daten über das File, das über Fortran I/O Dateinummer `Nummer` geöffnet ist und legt diese im `SFeld` ab. Die Werte in diesem Feld werden aus der `stat` Struktur, wie sie von `fstat(2)` zurückgegeben wird, wie folgt extrahiert: SFeld(1) Gerätenummer, SFeld(2) Inode-Nummer, SFeld(3) Dateizugriffsrechte, SFeld(4) Zahl der Verweise, SFeld(5) Besitzer uid, SFeld(6) Besitzer gid, SFeld(7) Gerätetyp, SFeld(8) Dateigrösse, SFeld(9) Zugriffszeit, SFeld(10) Modifikationszeit, SFeld(11) Änderungszeit, SFeld(12) Blockgrösse, SFeld(13) Zahl der angelegten Blöcke.

FDATE

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

Gibt das aktuelle Datum und die aktuelle Zeit wie folgt zurück: Tag Monat dd hh:mm:ss yyyy.

FTELL

```
INTEGER FUNCTION ftell(Nummer)

    INTEGER, INTENT(IN) :: Nummer

END FUNCTION ftell
```

Gibt den momentanen Abstand vom Anfang der Fortran-Datei mit der Dateinummer `Nummer` zurück oder -1 falls die Dateinummer nicht verbunden ist.

GAMMA

```
REAL FUNCTION gamma(x)

    REAL, INTENT(IN) :: x

END FUNCTION gamma
```

Gibt eine Näherung für $\Gamma(x)$ zurück. GAMMA ist eine generische Funktion, die jeden numerischen Gleitkommazahl-Typ akzeptiert.

GETCWD

```
INTEGER FUNCTION getcwd(Verzeichnis)

    CHARACTER(LEN=*), INTENT(OUT) :: Verzeichnis

END FUNCTION
```

Setzt `Verzeichnis` auf das aktuelle Arbeitsverzeichnis. Rückgabewert ist ungleich Null, wenn ein Fehler auftritt.

GETGID

```
INTEGER FUNCTION getgid()

END FUNCTION getgid
```

Gibt die Gruppen-Identifikationsnummer für den laufenden Prozess zurück.

GETPID

```
INTEGER FUNCTION getpid()

END FUNCTION getpid
```

Gibt die Prozess-Identifikationsnummer für den laufenden Prozess zurück.

GETUID

```
INTEGER FUNCTION getuid()

END FUNCTION getuid
```

Gibt die Benutzer-Identifikationsnummer zurück.

HOSTNM

```
INTEGER FUNCTION hostnm(Name)

    CHARACTER(LEN=*), INTENT(OUT) :: Name

END FUNCTION hostnm
```

Setzt `Name` auf den System-Namen. Rückgabewert ungleich Null im Fehlerfall.

IARGC

```
INTEGER FUNCTION iargc()

END FUNCTION iargc
```

Gibt die Anzahl der Kommandozeilen-Argumente zurück (ohne den Programmnamen).

ISATTY

```
LOGICAL FUNCTION isatty(Nummer)

    INTEGER, INTENT(IN) :: Nummer

END FUNCTION isatty
```

Gibt `.true.` genau dann zurück, wenn die Fortran I/O - `Nummer` einem Terminal zugeordnet ist.

ISNAN

```
LOGICAL FUNCTION isnan(x)

    REAL, INTENT(IN) :: x

END FUNCTION isnan
```

Gibt `.true.` zurück, wenn x NaN (N̈ot-a-Number) ist. Dies ist eine gernerische Funktion.

LINK

```
INTEGER FUNCTION link(Pfad1, Pfad2)

    CHARACTER(LEN=*), INTENT(IN) :: Pfad1, Pfad2

END FUNCTION link
```

Erzeugt einen (harten) Verweis von `Pfad1` auf `Pfad2`.

LNBLNK

```
INTEGER FUNCTION lnblnk(String)

    CHARACTER(LEN=*), INTENT(IN) :: String

END FUNCTION lnblnk
```

Anderer Name für die `len_trim` Standardfunktion. Gibt den Index des letzten Zeichens im `String` zurück, das nicht Leerzeichen ist.

LSTAT

```
INTEGER FUNCTION LSTAT(Datei, SFeld)

    CHARACTER(LEN=*), INTENT(IN)  :: Datei

    INTEGER         , INTENT(OUT) :: SFeld(13)

END FUNCTION LSTAT
```

Wenn `Datei` ein symbolischer Verweis ist, werden die Daten des Verweises selbst zurückgegeben. Siehe `FSTAT()` für weitere Einzelheiten. Gibt einen Wert ungleich Null im Fehlerfall zurück.

RAND

```
REAL FUNCTION rand(x)

    INTEGER, OPTIONAL, INTENT(IN) :: x

END FUNCTION rand
```

Gibt eine gleichverteilte Pseudo-Zufallszahl zurück, für die $0 \leq$ `rand` < 1 gilt. Wenn `x` 0 ist, wird die nächste Zahl der Sequenz zurückgegeben. Wenn `x` ist, wird der Generator mit einem Aufruf von `srand(0)` neu gestartet. Wenn x

einen anderen Wert hat, wird dieser als Startwert mittels `srand` verwendet.

SECNDS

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

Gibt die lokale Zeit in Sekunden seit Mitternacht abzüglich des Werts von `t` zurück. Dies ist eine generische Funktion.

SIGNAL

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

Schnittstelle zur Unix `signal`-Systemfunktion. Gibt ungleich 0 im Fehlerfall zurück.

SIZEOF

```
INTEGER FUNCTION sizeof(Objekt)
END FUNCTION sizeof
```

Das Argument `Objekt` ist der Name eines Ausdrucks oder Typs. Gibt die Grösse des `Objekt`s in Byte zurück.

STAT

```
 INTEGER FUNCTION stat(Datei, SFeld)
     CHARACTER(LEN=*), INTENT(IN)  :: Datei
     INTEGER         , INTENT(OUT) :: SFeld(13), Status
 END FUNCTION stat
```

Ermittelt Daten über die angegebene `Datei` und stellt sie im `SFeld` zur Verfügung. Siehe `FSTAT()` für weitere Einzelheiten. Gibt ungleich Null im Fehlerfall zurück.

SYSTEM

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

Rufe ein externes `Kommando` auf. Gibt den System-Exit-Code des Kommandos zurück.

TIME

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

Gibt die aktuelle Zeit wie in der UNIX `time` Funktion als Integer kodiert zurück.

UNLINK

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

Lösche `Datei`. Rückgabewert ungleich Null im Fehlerfall.

%VAL()
Wenn dies auf eine Variable in einer Liste formaler Argumente angewendet wird, wird diese Variable per Wert übergeben. Diese Pseudo-Funktion wird nicht empfohlen, und ist nur aus Kompatibilitätsgründen implementiert. Das F2003 `VALUE` Attribut ist der Standardmechanismus, um dies zu erreichen.

%REF()
Wenn dies auf eine Variable in einer Liste formaler Argumente angewendet wird, wird diese Variable per Referenz übergeben.

G95 Erweiterungen zu Standard-Unterprogrammen

ABORT

```
SUBROUTINE abort()

END SUBROUTINE abort
```

Das Programm bricht sich selbst über einen `SIGABORT` mit einem Speicherabzug ab (Unix).

CHDIR

```
SUBROUTINE chdir(Verzeichnis)

    CHARACTER(LEN=*), INTENT(IN) :: Verzeichnis

END SUBROUTINE
```

Setzt das aktuelle Arbeitsverzeichnis auf `Verzeichnis`.

DTIME

```
SUBROUTINE dtime(TFeld, Summe)

    REAL, OPTIONAL, INTENT(OUT) :: TFeld(2), Summe

END SUBROUTINE dtime
```

Setzt `TFeld(1)` gleich der Anzahl verstrichener Sekunden der Benutzer-Zeit des laufenden Prozesses seit dem letzten Aufruf von `DTIME`. Setzt `TFeld(2)` gleich der Anzahl verstrichener Sekunden der System-Zeit des laufenden Prozesses seit dem letzten Aufruf von `DTIME`. Setzt die `Summe` beider Zeiten.

ETIME

```
SUBROUTINE etime(TFeld, Summe)

    REAL, OPTIONAL, INTENT(OUT) :: TFeld(2), Summe

END SUBROUTINE etime
```

Setzt `TFeld(1)` gleich der Anzahl verstrichener Sekunden der Benutzer-Zeit des laufenden Prozesses. Setzt `TFeld(2)` gleich der Anzahl verstrichener Sekunden der System-Zeit des laufenden Prozesses. Setzt die `Summe` beider Zeiten.

EXIT

```
SUBROUTINE exit(Code)

    INTEGER, OPTIONAL, INTENT(IN) :: Code

END SUBROUTINE exit
```

Beendet das Programm mit Exit-`Code` nach Schliessen aller offenen Fortran I/O Dateien. Diese Subroutine ist generisch.

FDATE

```
SUBROUTINE fdate(Datum_Zeit)

    CHARACTER(LEN=*), INTENT(OUT) :: Datum_Zeit

END SUBROUTINE fdate
```

Setzt `Datum_Zeit` auf Datum und die aktuelle Zeit wie folgt: Tag Monat dd hh:mm:ss yyyy.

FLUSH

```
SUBROUTINE flush(Nummer)

    INTEGER, INTENT(IN) :: Nummer

END SUBROUTINE flush
```

Leert den Puffer der aktuell für Ausgabe geöffneten Fortran Datei-`Nummer`.

FSTAT

```
SUBROUTINE FSTAT(Nummer, SFeld, Status)

    INTEGER, INTENT(IN)  :: Nummer

    INTEGER, INTENT(OUT) :: SFeld(13), Status

END SUBROUTINE fstat
```

Ermittelt Daten über das File, das über Fortran I/O Dateinummer `Nummer` geöffnet ist und legt diese im `SFeld` ab. Setzt `Status` ungleich Null im Fehlerfall. Siehe `fstat` Funktion für weitere Informationen, wie `SFeld` besetzt wird.

GETARG

```
SUBROUTINE getarg(n, Wert)

    INTEGER         , INTENT(IN)  :: n

    CHARACTER(LEN=*), INTENT(OUT) :: Wert

END SUBROUTINE
```

Setzt `Wert` auf das `n`te Kommandozeilen-Argument.

GETENV

```
SUBROUTINE getenv(Variable, Wert)

    CHARACTER(LEN=*), INTENT(IN)  :: Variable

    CHARACTER(LEN=*), INTENT(OUT) :: Wert

END SUBROUTINE getenv
```

Fragt die Umgebungs-`Variable` ab und besetzt `Wert` mit ihrem Inhalt.

GETLOG

```
SUBROUTINE getlog(Name)

    CHARACTER(LEN=*), INTENT(OUT) :: Name

END SUBROUTINE getlog
```

Legt den zum Prozess gehörigen Login-Namen in `Name` ab.

IDATE

```
SUBROUTINE idate(m, t, j)

    INTEGER :: m, t, j

END SUBROUTINE idate
```

Setzt `m` auf den aktuellen Monat, `t` auf den aktuellen Tag und `j` auf das akuelle Jahr. Diese Routine ist zwischen verschiedenen Implementierungen nicht besonders portabel. Benutzen Sie die Standardprozedur `DATE_AND_TIME` für neuen Code.

LSTAT

```
SUBROUTINE lstat(Datei,SFeld,Status)

    CHARACTER(LEN=*), INTENT(IN)  :: Datei

    INTEGER         , INTENT(OUT) :: SFeld(13), Status

END SUBROUTINE lstat
```

Wenn `Datei` ein symbolischer Verweis ist, werden die Daten des Verweises selbst zurückgegeben. Siehe `FSTAT()` für weitere Einzelheiten.

RENAME

```
SUBROUTINE rename(Pfad1, Pfad2, Status)

    CHARACTER(LEN=*) , INTENT(IN)  :: Pfad1, Pfad2

    INTEGER, OPTIONAL, INTENT(OUT) :: Status

END SUBROUTINE rename
```

Ändert Datei-`Pfad1` in `Pfad2`. Falls angegeben, setzt Status auf ungleich Null im Fehlerfall.

SIGNAL

```
SUBROUTINE signal(Signal, Handler, Status)

    INTEGER  , INTENT(IN)  :: Signal

    PROCEDURE, INTENT(IN)  :: Handler

    INTEGER  , INTENT(OUT) :: Status

END SUBROUTINE signal
```

Schnittstelle zur Unix `signal`-Systemroutine. Setzt `Status` auf ungleich Null im Fehlerfall.

SLEEP

```
SUBROUTINE sleep(Sekunden)

    INTEGER, INTENT(IN) :: Sekunden

END SUBROUTINE sleep
```

Veranlasst den Prozess, für die angegebene Zahl von `Sekunden` zu pausieren.

SRAND

```
SUBROUTINE srand(seed)

    INTEGER, INTENT(IN) :: seed

END SUBROUTINE srand
```

Startet den Zufallsgenerator neu. Siehe `rand()` Funktion für weitere Einzelheiten.

STAT

```
SUBROUTINE stat(Datei, SFeld, Status)

    CHARACTER(LEN=*), INTENT(IN)  :: Datei

    INTEGER         , INTENT(OUT) :: SFeld(13), Status

END SUBROUTINE
```

Ermittelt Daten über die angegebene `Datei` und legt sie im `SFeld` ab. Siehe `fstat()` für weitere Einzelheiten. Setzt `Status` auf ungleich Null im Fehlerfall.

SYSTEM

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

Rufe ein externes `Kommando` auf. Falls angegeben, wird `Code` auf den System-Exit-Code des Kommandos gesetzt.

UNLINK

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

Lösche `Datei`. Rückgabewert in `Status` ungleich Null im Fehlerfall.