G95

It's free crunch time – Ici, on compile gratis

http://www.g95.org

Auteurs du manuel :
Doug Cox
Andy Vaught

Traduction en français :
Siegfried MEUNIER-GUTTIN-CLUZEL

Caractéristiques principales de G95

- Un compilateur gratuit à la norme Fortran 95.
- La version actuelle (octobre 2008) de g95 est la 0.92.
- Licence GNU Open Source, GPL.
- Le fonctionnement des programmes compilés peut être modifié par de nombreuses variables d'environnement, documentées dans le programme compilé lui-même.
- TR15581– Tableaux dynamiques en arguments muets, type structure dérivé.
- F2003 – pointeurs de procédures, constructeurs de structures, interopérabilité
- Procédures intrinsèques et modules du F2003.
- Les arguments formels de type `VALUE` dans les procédures sont passés par valeur.
- Option virgule pour OPEN, READ, and WRITE pour préciser le séparateur décimal.
- Des crochets `[` et `]` peuvent être employés pour les constructeurs de tableaux.
- Instruction `IMPORT`, utilisée dans le corps d'une interface pour permettre l'accès à des entités de l'unité de programme hôte.
- `MIN()` et `MAX()` pour les types caractères comme pour les types numériques.
- `OPEN` Pour des E/S de type flots ou "Transparentes".
- Compatibilité ascendante avec l'Application Binary Interface (ABI) du g77.
- Entiers par défaut de 32 ou 64 bits disponibles.
- Invoque la commande `SYSTEM()` .
- Sources avec des tabulations autorisées.
- Option autorisant les $ dans les noms symboliques.
- Données Hollerith.
- Extension DOUBLE COMPLEX.
- Longueur variable pour les `COMMON` nommés.
- Mélange nombres et caractères dans les `COMMON` et les `EQUIVALENCE`.
- `INTEGER` variantes : 1, 2, 4, 8.
- `LOGICAL` variantes : 1, 2, 4, 8.
- `REAL` variantes : 4, 8, (16 est expérimental
- `REAL(KIND=10)` pour les systèmes compatibles x86. 19 chiffres de précision, domaine couvert $10^{\pm 4931}$.
- Les instructions d'écritures de réels formattées par liste affichent le nombre minimal de chiffres nécessaires pour distinguer le nombre de façon unique.
- Lignes de déboguage style VAX (D).
- Option constantes caractères style C (e.g. 'hello\nworld').
- Descripteurs d'édition \ et $.
- Fonctions système intrinsèques style VAX (SECNDS etc.)
- Bibliothèque d'extensions du système unix (getenv, etime, stat, etc.)
- Détecter les tableaux non compatibles ou non alloués à l'exécution - voir Table IV sur : `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`
- Détection des fuites de mémoire - voir Table V sur: `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`
- Localisation de erreurs l'exécution.
- Compilation intelligente pour éviter les cascades de compilations de modules.
- Option de compatibilité avec F. Voir `http://www.fortran.com/F`. G95 peut être construit comme un compilateur F.
- Possibilité d'interrompre/reprendre le programme disponible pour x86/Linux.
- Les compteurs de boucles obsolètes en réels exigent une compilation avec `-freal-loops`.
- La réponse du développeur à un rapport de bug est d'habitude assez rapide.
- Compatible avec les versions 4.0.3 à 4.1.2 de GCC.
- Disponible pour Linux/x86, PowerPC, 64-bit Opteron, 64-bit Itanium, 64-bit Alpha.
- Disponible pour Windows/Cygwin, MinGW & Interix.
- Disponible pour OSX sur Power Mac G4, x86-OSX.
- Disponible pour FreeBSD sur x86, HP-UX 11, Sparc-Solaris, x86-Solaris, OpenBSD, NetBSD, AIX, IRIX, Tru64 UNIX sur Alpha.

- Version Fink aussi disponible.
- Les binaires de la version 'stable' et des versions courantes sont disponibles pour la plupart des plateformes à `http://ftp.g95.org`.
- Accepte l'option `CONVERT=` dans une instruction OPEN pour préciser une conversion de boutisme. Les valeurs acceptées sont 'big_endian', 'little_endian', 'native' et 'swap'.

De temps en temps, je rencontre quelqu'un avec qui j'ai échangé des emails au sujet de g95. Le commentaire le plus fréquent que j'entends dans ces situations est quel boulot extraordinaire je fais tout seul. J'en ris tout le temps et je fais remarquer que je ne l'ai jamais fait seul. Le nombre de personnes qui ont aidé activement au g95 est probablement proche d'un millier. On croit que la personne qui écrit le code fait tout le travail, alors qu'en réalité les personnes qui distillent un plantage en une dizaine de lignes rendent un très précieux service, qui est souvent oublié. Ecrire quelquechose d'aussi compliqué qu'un compilateur fortran moderne n'est pas quelque chose que l'on fait seul. Je le sais.

Comme la plupart des choses, g95 est né d'une frustration. J'écrivais le code de ma thèse de doctorat en fortran 77 en utilisant g77. Fortran est vraiment un formidable langage pour le calcul numérique – c'est un langage rapide et rustique pour les gens qui s'intéressent plus au résultat qu'à l'écriture du programme. Le code de ma thèse contenait beaucoup de structures de données plutôt sophistiquées – listes chaînées, arbres, matrices creuses, permettant la génération de maillages d'éléments finis, la résolution d'équations de Poisson, d'expansion en multipôles, de minimisation par gradients conjugués et pas mal de geométrie analytique. Comme j'utilisais fortran 77, le code a fini par être un gros bricolage qui aurait énormément bénéficié de l'allocation dynamique de mémoire et des types dérivés. En plus ma thèse se terminait et j'avais besoin d'un nouveau défi.

Au delà du confort des fonctionalités plus avancées du langage, j'ai aussi été grandement inspiré par le travail de Bill Kahan. Ce que j'ai retenu après avoir lu plusieurs articles de Bill était l'idée que bien que les calculs numériques étaient délicats, on pouvait trouver des façons de faire telles que les erreurs soient réduites au point que plus personne n'y fait attention. A ce point l'utilisateur est souvent à la merci de l'auteur de la bibliothèque.

Bien que le compilateur soit la partie sympa, ce sont les bibliothèques qui m'ont toujours intéressé le plus. Les actions du compilateur sont définies de façon plutôt stricte par la norme, et c'est dans la bibliothèque que l'on peut innover et expérimenter à loisir. Même quand il était dans un état encore primitif, il y avais déjà plus d'accessoires dans la bibliothèque que chez les autre fournisseurs. La possibilité de reprise à partir d'un fichier image est une chose que j'ai voulu pendant des années avant d'avoir la chance de pouvoir l'implémenter.

Cela a été une grande joie d'écrire g95 et j'espère en assurer la maintenance dans les décennies qui viennent.

Andy Vaught
Mesa, Arizona
Octobre 2006

License

G95 lui-même est sous licence GNU General Public Licence (GPL). Pour tous les détails légaux, voir `http://www.gnu.org/licenses/gpl.html`.

La bibliothèque d'exécution (runtime library) est presque entièrement GPL mais contient une exception à la GPL qui permet à l'utilisateur de lier les bibliothèques g95 à des codes qui ne sont pas couverts par la GPL et de distribuer l'exécutable sans qu'il soit couvert par la GPL, ou qu'il soit affecté par la GPL en quelque manière que ce soit.

Notes d'installation

Unix (Linux/OSX/Solaris/Irix/etc.):
Ouvrez un terminal et allez dans le répertoire dans lequel vous voulez installer g95. Pour télécharger g95 et l'installer, lancez la commande suivante :

```
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
```

Les répertoires et les fichiers suivants devraient être présents :

```
./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
```

Le fichier `cc1` est un lien symbolique vers `f951` dans le même répertoire.

Cygwin

L'option `-mno-cygwin` permet à la version Cygwin de g95 de construire des exécutables qui n'ont pas besoin d'accéder au fichier cygwin1.dll pour fonctionner, et donc peuvent facilement être lancés sur d'autres systèmes. En plus les exécutables sont libres des restrictions attachées à la licence GNU GPL. Pour installer une version Cygwin avec une option -mno-cygwin fonctionnelle vous aurez besoin d'avoir les bibliothèques mingw installées, celle-ci sont disponibles sur le site Cygwin à l'adresse `http://www.cygwin.com`.
Téléchargez les binaires à partir de `http://ftp.g95.org/g95-x86-cygwin.tgz` vers votre répertoire Cygwin (d'habitude c'est c:\Cygwin). Lancez une session Cygwin et tapez la commande :

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

Ceci installera l'exécutable de g95 dans le répertoire `/usr/local/bin`. Attention, n'utilisez pas Winzip pour extraire les fichiers du tarball sinon les liens indispensables ne seront pas créés correctement.

MinGW

Les binaires de g95 pour l'environnment MS-Windows sont des installateurs auto-extractibles. Deux versions sont actuellement disponibles. Les utilisateurs de Windows 98 devraient utiliser le paquetage construit avec gcc 4.0.4 disponible à `http://ftp.g95.org/g95-MinGW.exe`. Les utilisateurs de Windows NT, XP et 2000 ont le choix soit d'utiliser le même paquetage ou un autre construit avec gcc 4.1.2, disponible à `http://ftp.g95.org/g95-MinGW-41.exe`.

Le système libre MinGW/Msys fournit les fichiers GNU GCC dont a besoin g95, ce qui inclue `ld.exe` (l'éditeur de liens), et `as.exe` (l'assembleur GNU) venant du paquetage binutils, disponible à `http://www.mingw.org`. Le script d'installation gère deux types d'installations. Si MinGW n'est pas trouvé, il installe g95 avec quelque programmes et bibliothèques essentiels du binutils de MinGW dans un répertoire choisi par l'utilisateur. Ajoutez le répertoire d'installation dans votre `PATH`, et faites que votre variable d'environnement `LIBRARY_PATH` pointe sur votre répertoire d'installation.

Si vous installez le paquetage g95-MinGW-41.exe, (qui ne marchera pas sous Windows 98 et Windows ME), initialisez la variable d'environnement `G95_LIBRARY_PATH` à la place

`G95_LIBRARY_PATH=`*path-to-MinGW*`/lib`

Le paquetage g95-MinGW-41.exe n'entrera pas en conflit avec gfortran ou gcc.

Si MinGW est déjà installé sur votre système, il est recommandé d'installer g95 dans le répertoire racine de MinGW, (généralement `C:\mingw`) pour éviter d'éventuels conflits. Si l'installateur détecte MinGW, il essaie d'installer g95 dans le système de fichiers de MinGW. Ajoutez le répertoire `MinGW\bin` dans votre `PATH` et modifiez votre variable d'environnement

`LIBRARY_PATH=`*path-to-MinGW*`/lib`

Sous Windows 98 et Windows ME ceci demande généralement d'éditer le fichier système `autoexec.bat` et il faut un redémarrage pour que les changements soient pris en compte.

Note pour les utilisateurs de Windows XP : MinGW permet actuellement seulement 8 mégaoctets pour la pile. Si votre application a besoin d'accéder à plus de mémoire, essayez de compiler avec `-Wl,--heap=0x01000000`. Utilisez des valeurs hexadécimales plus grandes pour `--heap` jusqu'à ce que votre programme fonctionne.

Lancer G95

G95 détermine comment un fichier doit être compilé en se basant sur son extension. Les extensions de nom de fichier autorisées pour un fichier source Fortran sont limitées à `.f`, `.F`, `.for`, `.FOR`, `.f90`, `.F90`, `.f95`, `.F95`, `.f03` et `.F03`. L'extension du nom de fichier détermine si la source Fortran sera considérée comme étant en format fixe ou en format libre. Les fichiers se terminant en `.f`, `.F`, `.for`, et `.FOR` sont supposés être des sources en format fixe compatible avec les anciens fichiers f77. Les fichiers se terminant en `.f90`, `.F90`, `.f95`, `.F95`, `.f03` and `.F03` sont supposés être des sources en format libre. Les fichiers se terminant en lettres majuscules sont pré-traités par le préprocesseur C par défaut, les fichier se terminant par des lettres minucules ne sont pas pré-traités par défaut.

Les options de base pour compiler des sources Fortran avec g95 sont :

`-c` Compile seulement, ne lance pas l'éditeur de liens.

`-v` Affiche les programmes réellement appelés par g95 et leurs arguments. Particulièrement utile pour déméler les problèmes de chemins.

`-o` Précise le nom du fichier de sortie, soit un fichier objet, soit un exécutable. Une extension `.exe` est ajoutée automatiquement sous Windows. Si on ne spécifie pas de fichier de sortie, le fichier de sortie par défaut est nommé `a.out` sous unix, ou `a.exe` sous un système Windows.

Exemples simples:

```
g95 -c hello.f90
```
Compile `hello.f90` en un fichier objet nommé `hello.o`.

```
g95 hello.f90
```

Compile `hello.f90` et effectue l'edition de lien pour produire un executable `a.out` (sous unix ou linux), ou `a.exe` (sous les systèmes MS Windows).

`g95 -c h1.f90 h2.f90 h3.f90`
Compile plusieurs fichiers source. Si tout va bien, les fichiers objets `h1.o`, `h2.o` et `h3.o` sont créés.

`g95 -o hello h1.f90 h2.f90 h3.f90`
Compile plusieurs fichiers source et effectue l'édition de liens pour créer un fichier exécutable nommé `hello` sous unix, ou `hello.exe` sous un système MS Windows.

Résumé des options

`g95 [ -c | -S | -E ]`	Compile & assemble | Produit le code assembleur | Listing du code source
`[-g] [-pg]`	Options de Débogage
`[-O[`*n*`] ]`	Niveaux d'Optimisation, $n = 0, 1, 2, 3$
`[-s ]`	Supprime les infos de debogage
`[-W`*warn*` ] [-pedantic]`	Active les avertissements
`[-I`*dir*` ]`	Répertoire à parcourir pour les Inclusions
`[-L`*dir*` ]`	Répertoire à parcourir pour les Librairies
`[-D macro[=valeur]... ]`	Definit une macro
`[-U macro ]`	Neutralise une macro
`[-f option ...]`	Options générales de compilation
`[-m option-machine ...]`	Options liées aux machines. Voir le manuel de GCC
`[-o fichier-sortie ]`	Nom du fichier de sortie
`fichier-entrée`	

Les Options de G95

Usage: g95 [options] fichier...

`-pass-exit-codes`	Sort avec le code d'erreur le plus élevé d'une phase.
`--help`	Affiche ces informations.
`--target-help`	Affiche les options de ligne de commande spécifiques à une architecture. (Utiliser '`-v --help`' pour afficher les options de ligne de commande des sous-processus).
`-dumpspecs`	Affiche toutes les chaînes de spcifications.
`-dumpversion`	Affiche la version du compilateur.
`-dumpmachine`	Affiche le processeur cible du compilateur.
`-print-search-dirs`	Affiche les répertoires du chemin de recherche du compilateur.
`-print-libgcc-file-name`	Affiche le nom de la bibilothèque attachée au compilateur.
`-print-file-name=`*lib*	Affiche le chemin complet jusqu'à la bibliothèque *lib*.
`-print-prog-name=`*prog*	Affiche le chemin complet jusqu'au composant *prog* du compilateur.
`-print-multi-directory`	Affiche le répertoire racine pour les versions de libgcc.
`-print-multi-lib`	Affiche la correspondance entre les options de ligne de commande et les multiples chemins de recherche des bibliothèques.
`-print-multi-os-directory`	Affiche le chemin relatif vers les bibliothèques du système d'exploitation.
`-Wa,`*options*	Passe les *options* séparées par des virgules à l'assembleur.
`-Wp,`*options*	Passe les *options* séparées par des virgules au préprocesseur.
`-Wl,`*options*	Passe les *options* séparées par des virgules à l'éditeur de liens.
`-Xassembler` *arg*	Passe *arg* à l'assembleur.
`-Xpreprocessor` *arg*	Passe *arg* au préprocesseur.
`-Xlinker` *arg*	Passe *arg* à l'éditeur de liens.
`-save-temps`	Ne pas détruire les fichiers temporaires.
`-pipe`	Utiliser des tuyaux (pipes) plutôt que des fichiers temporaires.
`-time`	Chonomètre les sous-processus. Non disponible sur certaines plateformes (MinGW, OSX).

`-specs=`*file*	Remplace la configuration incorporée par le contenu du fichier *file*.
`-std=`*standard*	Supposer que les fichier d'entrée suivent le *standard*.
`-B` *directory*	Ajoute *directory* aux chemins de recherche du compilateur.
`-b` *machine*	Exécute gcc pour l'architecture *machine*, si elle est installée.
`-V` *version*	Exécute gcc version nummméro *version*, si il est installé.
`-v`	Affiche les programmes appelés par le compilateur
`-M`	Produit les lignes de dépendance type Makefile sur la sortie standard.
`-###`	Comme `-v` mais les options sont indiquées et les commandes ne sont pas exécutées.
`-E`	Prétraitement seulement; ne pas compiler, assembler ou éditer les liens.
`-S`	Compile seulement, ne pas assembler ou éditer les liens.
`-c`	Compile et assemble, mais n'édite pas les liens.
`-o` *file*	Place la sortie dans le fichier *file*.
`-x` *langage*	Specifie le *langage* des fichiers d'entrée qui suivent. Les langages possibles incluent : c, c++, assembleur, aucun; 'none' signifie revenir au comportement par défaut qui consiste à deviner le langage en se basant sur l'extension du fichier.

Les options qui commencent par `-g`, `-f`, `-m`, `-O`, `-W`, ou `--param` sont automatiquement passées aux sous-processus variés que g95 appelle. Pour passer d'autres options à ces processus on doit utiliser les options commençant avec la lettre `-W`*letter*. Pour des instructions pour signaler des bugs, veuillez voir : `http://www.g95.org`.

Par défaut, les programmes compilés avec g95 ne sont pas optimisés. Le n dans `-O`n précise le niveau d'optimisation, de 0 à 3. Zèro signifie pas d'optimisation, et les nombres plus élevés impliquent une optimisation plus agressive. Demander une optimisation donne le droit au compilateur de changer le code pour le rendre plus rapide. Le résultat des calculs est souvent affecté de façon subtile. Utiliser `-O` est la même chose que `-O1`.

Des gains de vitesse significatifs peuvent être obtenus en précisant au moins `-O2 -march=`*arch* où *arch* est l'architecture de votre processeur, ie `pentium4`, `athlon`, `opteron`, etc. Les autre options Fortran typiques sont `-funroll-loops`, `-fomit-frame-pointer`, `-malign-double` et `-msse2`. Pour des informations sur toutes les options de GCC disponibles quand on compile avec g95, voir
`http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc`.

Les options du préprocesseur

G95 peut traiter des fichiers contenant des constructions du préprocesseur C.

`-cpp`	Force le traitement des fichiers d'entrée par le préprocesseeur C
`-no-cpp`	Empêche le prétraitement des fichiers d'entrée
`-D name[=valeur]`	Définit une macro du préprocesseur
`-U name`	Neutralise une macro du préprocesseur
`-E`	Montre seulement la source prétraitée
`-I` *repertoire*	Ajoute *repertoire* au chemin de recherche des fichier à inclure et des modules. Les fichiers sont recherchés dans plusieurs répertoire dans cet ordre : Le répertoire du fichier source principal, le répertoire courant, les répertoires spécifiés par `-I`, les répertoire spécifiés lans la variable d'environnement `G95_INCLUDE_PATH` et enfin dans les répertoires systèmes.

Les options Fortran

`-Wall`	Autorise la plupart des messages d'avertissement.
`-Werror`	Change les avertissements en erreurs.
`-Werror=`*numbers*	Change la liste d'avertissements séparés par des virgules en erreurs.
`-Wextra`	Autorise les avertissements non autorisés par `-Wall`. Ce sont `-Wobsolescent`, `-Wunused-module-vars`, `-Wunused-module-procs`, `-Wunused-internal-procs`, `-Wunused-parameter`, `-Wunused-types`, `-Wmissing-intent` and `-Wimplicit-interface`.

`-Wglobals`	Vérification croisée de la définition et de l'utilisation des procédures au sein d'un même fichier source. Activé par défaut, utiliser `-Wno-globals` pour la désactiver.
`-Wimplicit-none`	Identique à `-fimplicit-none`.
`-Wimplicit-interface`	Prévient de l'utilisation d'une interface implicite.
`-Wline-truncation`	Prévient d'une ligne source tronquée.
`-Wmissing-intent`	Warn about missing intents on format arguments.
`-Wobsolescent`	Prévient d'une construction obsolescente.
`-Wno=`*numbers*	Déactive une liste d'avertissements séparés par des virgules et identifiés par des numéros.
`-Wuninitialized`	Prévient si des variables sont utilisées avant d'avoir été initialisées. Nécessite l'option `-O2`.
`-Wunused-internal-procs`	Prévient si une procédure interne n'est jamais utilisée.
`-Wunused-vars`	Prévient de variables non utilisées.
`-Wunused-types`	Prévient si un type présent dans un module n'est jamais utilisé. N'est pas compris dans `-Wall`.
`-Wunset-vars`	Prévient de variables non initialisées.
`-Wunused-module-vars`	Prévient de variables de module non utilisées. Utile pour construire des clauses `ONLY`.
`-Wunused-module-procs`	Prévient de procédures de module non utilisées. Utile pour construire des clauses `ONLY`.
`-Wunused-parameter`	Prévient de paramètres non utilisés. Pas comprise dans `-Wall`.
`-Wprecision-loss`	Prévient d'une perte de précision dans une conversion de type implicite.
`-fbackslash`	Interprète les barres de fractions inverses (backslashes) dans les constantes de type caractère comme de codes d'échappement. Cette option est activée par défaut. Utiliser `-fno-backslash` pour traiter les barres de fractions inverses littéralement.
`-fc-binding`	Affiche les prototypes C des procédures sur la sortie standard.
`-fd-comment`	Rend les lignes commençant par un D exécutables en format fixe.
`-fdollar-ok`	Autorise le caractère dollar dans les identificateurs.
`-fendian=`*valeur*	Impose le boutisme (endian-ness) des lectures et écritures non formatées. *valeur* doit être `big` ou `little`. Est prioritaire par rapport aux variables d'environnement d'exécution.
`-ffixed-form`	Suppose que la source est en format fixe.
`-ffixed-line-length-132`	lignes de 132 caractères de large en format fixe.
`-ffixed-line-length-80`	lignes de 80 caractères de large en format fixe.
`-ffree-form`	Suppose que le fichier source est en format libre.
`-ffree-line-length-huge`	Permet des lignes très longues dans la source (10k).
`-fimplicit-none`	Précise qu'aucun typage implicite n'est permis, a moins de l'imposer explicitement par une instruction `IMPLICIT`.
`-fintrinsic-extensions`	Autorise des fonctions intrinsèques spécifiques au g95, même avec un mode `-std=` .
`-fintrinsic-extensions=`	Inclue la sélection de fonctions intrisèques, même en mode `-std=`. La liste est séparée par des virgules et insensible à la casse (majuscules - minuscules).
`-fmod=`*directory*	Met les fichier des modules dans le répertoire *directory*.
`-fmodule-private`	Met l'accessibilité par défaut des objets des modules à `PRIVATE`.
`-fmultiple-save`	Permet de spécifier l'attribut `SAVE` plusieurs fois.
`-fone-error`	Force l'arrêt de la compilation à la première erreur.
`-ftr15581`	Autorise les extensions TR15581 concernant les tableaux dynamiques, même avec les modes `-std=F` ou `-std=f95`.
`-std=F`	Prévient de caractéristiques non-F. Voir `http://www.fortran.com/F`.
`-std=f2003`	Vérification Fortran 2003 stricte.

`-std=f95`	Vérification Fortran 95 stricte.
`-i4`	Mets la variante des entiers sans spécification à kind=4 (32 bits).
`-i8`	Mets la variante des entiers sans spécification à kind=8 (64 bits).
`-r4`	Mets la variante des réels sans spécification à kind=4.
`-r8`	Mets la variante des réels sans spécification à double precision.
`-r16`	Mets la variante des réels sans spécification à kind=16.
`-d8`	Implique `-i8` et `-r8`.

Options de génération du code

`-fbounds-check`	Vérifie les limites des tableaux et des chaînes de caractères pendant l'exécution.
`-fcase-upper`	Met en majuscule tous les symboles publics.
`-fleading-underscore`	Ajoute un caractère souligné à tous les noms publics.
`-fonetrip`	Exécute au moins une fois les boucles `DO`. (bug du `FORTRAN` 66).
`-fpack-derived`	Essaie de ranger les types dérivés de façon la plus compacte possible. Nécessite moins de mémoire, mais peut être plus lent.
`-freal-loops`	Autorise les compteurs rels dans les boucles DO.
`-fqkind=`*n*	Met la variante d'un réel avec l'exposant 'q' à *n*.
`-fsecond-underscore`	Ajoute un second caractère souligné final aux noms qui en ont déjà un (par défaut). Utiliser `-fno-second-underscore` pour le supprimer.
`-fshort-circuit`	fait que les opérateurs `.AND.` et `.OR.` ne calculent pas le second opérande si la valeur de l'expression est connue grâce au premier opérande.
`-fsloppy-char`	Supprime les erreurs quand on écrit des données non-caractères dans des variables caractère et permet les comparaisons entre des variables INTEGER et CHARACTER.
`-fstatic`	Met les variables locales en mémoire statique quand c'est possible. Ce n'est pas la même chose que de faire une édition de liens statique (`-static`).
`-ftrace=`	`-ftrace=frame` Va ajouter du code pour permettre de remonter l'historique de la pile en cas de fin anormale du programme. Cela va ralentir votre programme. `-ftrace=full` permet en plus de trouver le numéro de la ligne où s'est produite une exception arythmétique (encore plus lent). Le défaut est `-ftrace=none`.
`-funderscoring`	Ajoute un caractère souligné final aux noms globaux. Cette option est activée par défaut, utiliser `-fno-underscoring` pour la désactiver.
`-max-frame-size=`*n*	La taille en octets que doit atteindre un seul bloc de pile avant que les tableaux soient allouée dynamiquement.
`-finteger=`*n*	Initialise les variables scalaires entières non initialisées à *n*.
`-flogical=`*valeur*	Initialise les variables scalaires logiques non initialisées à *valeur*. Les *valeur* légales sont `none`, `true` et `false`.
`-freal=`*valeur*	Initialise les variables scalaires réelles et complexes non initialisées Les *valeur* légales sont `none`, `zero`, `nan`, `inf`, `+inf` et `-inf`.
`-fpointer=`*valeur*	Initialise les pointeurs scalaires. Les *valeurs* légales sont `none`, `null` et `invalid`.
`-fround=`*valeur*	Contrôle les arrondis à l'exécution. *valeur* peut être `nearest`, `plus`, `minus` et `zero`. Le comportement par défaut est d'arrondir à la valeur la plus proche (nearest), `plus` arrondi vers plus l'infini, `minus` arrondi vers moins l'infini, `zero` arrondi vers zéro.
`-fzero`	Initialise les types numériques à zéro, les variables logiques à faux et les pointeurs à nul. Les autres options d'initialisations remplacent celle-ci.

Les options de répertoires

`-I` *repertoire*	Ajoute *repertoire* au chemin des recherche des fichiers à inclure et des modules.
`-L`*repertoire*	Ajoute *repertoire* au chemin de recherche des bibliothèques.
`-fmod=`*repertoire*	Met les fichiers de modules dans *repertoire*

Les Variables d'Environment

L'environnement d'exécution du g95 fournit plusieurs options pour ajuster le comportement de votre programme quand il tourne. Elles sont contrôlables grâce à de variables d'environnement. Faire tourner un programme compilé avec g95 en ajoutant l'option `--g95` va écrire toutes ces options sur la sortie standard. Les valeurs des différentes variables d'environnement sont toujours des chaînes de caractères, mais celle-ci sont interprétées comme des entiers ou des valeurs logiques. Seul le premier caractère des valeurs logiques est pris en compte et doit être '`t`', '`f`', '`y`', '`n`', '`1`' ou '`0`' (les majuscules sont aussi acceptées). Si une valeur n'est pas valide, aucune erreur n'est provoquée et le comportement par défaut est utilisé. Pour les variables d'environnement de GCC utilisées par g95, comme `LIBRARY_PATH`, voir la documentation de GCC.

`G95_STDIN_UNIT`	Entier	Numéro de l'unité qui sera préconnectée l'entrée standard. Pas de pré-connection si négatif, le défaut est 5.
`G95_STDOUT_UNIT`	Entier	Numéro de l'unité qui sera préconnectée la sortie standard. Pas de pré-connection si négatif, le défaut est 6.
`G95_STDERR_UNIT`	Entier	Numéro de l'unité qui sera préconnectée la sortie erreur. Pas de pré-connection si négatif, le défaut est 0.
`G95_USE_STDERR`	Logique	Envoie la sortie des bibliothèques vers la sortie erreur la place de la sortie standard. Le défaut est Yes.
`G95_ENDIAN`	Chaîne	Le format endian (boutisme) à utiliser pour les E/S de données non formattées. Les valeurs possibles sont `BIG`, `LITTLE` ou `NATIVE`. Le défaut est `NATIVE`.
`G95_CR`	Logique	Ecrire des retours chariot pour les enregistrement séquentiels formattés. Le défaut est vrai (TRUE) sous Windows/non-Cygwin (natif), et faux (FALSE) ailleurs.
`G95_INPUT_CR`	Logique	Traiter un retour chariot-passage la ligne comme un marqueur d'enregistrement plutôt que juste comme un passage la ligne. Le défaut est vrai (TRUE).
`G95_IGNORE_ENDFILE`	Logique	Ignore les tentatives de lire après l'enregistrement de fin de fichier (ENDFILE) en accès séquentiel. Le défaut est faux (FALSE).
`G95_TMPDIR`	Chaîne	Répertoire pour les fichier temporaires (scratch). Remplace la variable d'environnement `TMP`. Si `TMP` n'est pas définie on utilise `/var/tmp`. Pas de valeur par défaut.
`G95_UNBUFFERED_ALL`	Logique	Si vrai (TRUE), aucune sortie n'est tamponnée. Celà va ralentir les écritures importantes mais peut être utile pour forcer les données être affichées immédiatement. Le défaut est faux (FALSE).
`G95_SHOW_LOCUS`	Logique	Si vrai (TRUE), affiche le nom du fichier et le numéro de la ligne où une erreur d'exécution s'est produite. Le défaut est vrai (TRUE).
`G95_STOP_CODE`	Logique	If TRUE, stop codes are propagated to system exit codes. Default TRUE.
`G95_OPTIONAL_PLUS`	Logique	Ecrit des signes plus optionnels quand c'est permis. Le défaut est faux (FALSE).
`G95_DEFAULT_RECL`	Entier	Longueur maximum par défaut pour un enregistrement d'un fichier séquentiel. Très utile pour ajuster la longueur des lignes des unités pré-connectées. Le défaut est 50000000.
`G95_LIST_SEPARATOR`	Chaîne	Séparateur à utiliser en écriture dirigée par liste. Peut contenir n'importe quel nombre d'espaces et au plus une virgule. Le défaut est une espace simple.
`G95_LIST_EXP`	Entier	La dernière puissance de dix qui n'utilise pas le format exponentielle en écriture dirigée par liste. Le défaut est 6.
`G95_COMMA`	Logique	Utilise la virgule comme séparateur décimal par défaut pour les E/S. Le défaut est faux (FALSE).

G95_EXPAND_UNPRINTABLE	Logique	Pour les écritures formattées, écrire les caractères qui seraient autrement non imprimable avec des séquences commenant par \. Le défaut est faux (FALSE).
G95_QUIET	Logique	Supprime les caractères cloche (\a) dans les sortie formatées. Le défaut est faux (FALSE).
G95_SYSTEM_CLOCK	Entier	Nombre de tics par seconde que donne la fonction intrinsèque SYSTEM_CLOCK(). Zéro désactive l'horloge. Le défaut est 100000.
G95_SEED_RNG	Logique	si vrai (TRUE), initialise le générateur aléatoire avec une nouvelle graine quand le programme est lancé. Le défaut est faux (FALSE).
G95_MINUS_ZERO	Logique	Si vrai (TRUE), écrit les valeurs nulles sans signe moins dans le cas de sorties formattées (non dirigée par liste), même si la valeur interne est un zéro négatif ou moins zéro. C'est la façon traditionnelle d'écrire les zéros, mais pas celle de la norme. Le défaut est faux (FALSE).
G95_ABORT	Logique	Si vrai (TRUE), sauve un fichier image (core) en cas de fin anormale du programme. Utile pour trouver la localisation d'un problème. Le défaut est faux (FALSE).
G95_MEM_INIT	Chaîne	Comment initialiser la mémoire allouée. La valeur par défaut est NONE, pour ne pas faire d'initialisation (plus rapide), NAN pour un nombre dénormal (Not-A-Number) avec la mantisse 0x00f95 ou une valeur hexadécimale personalisée.
G95_MEM_SEGMENTS	Entier	Nombre maximum de segments de mémoire encore alloués afficher quand le programme s'arrête. 0 signifie en afficher aucun, un nombre négatif signifie les afficher tous. Le défaut est 25.
G95_MEM_MAXALLOC	Logique	Si vrai (TRUE), affiche le nombre maximum d'octets de mémoire utilisateur utilisés pendant l'exécution du programme. Faux par défaut.
G95_MEM_MXFAST	Entier	Maximum request size for handing requests in from fastbins. Fastbins are quicker but fragment more easily. Default 64 bytes.
G95_MEM_TRIM_THRESHOLD	Entier	Amount of top-most memory to keep around until it is returned to the operating system. -1 prevents returning memory to the system. Useful in long-lived programs. Default 262144.
G95_MEM_TOP_PAD	Entier	Espace mémoire supplémentaire à allouer quand on reçoit de la mémoire du système d'exploitation. Peut accélerer les demandes de mémoire futures. Le défaut est zéro.
G95_SIGHUP	Chaîne	Whether the program will IGNORE, ABORT, DUMP or DUMP-QUIT on SIGHUP. Default ABORT. Seulement sous Unix.
G95_SIGINT	Chaîne	Whether the program will IGNORE, ABORT, DUMP or DUMP-QUIT on SIGINT. Default ABORT. Seulement sous Unix.
G95_SIGQUIT	Chaîne	Whether the program will IGNORE, ABORT, DUMP or DUMP-QUIT on SIGQUIT. Default ABORT. Seulement sous Unix.
G95_CHECKPOINT	Entier	Sous x86 Linux, le nombre de secondes entre deux sauvegardes de fichier image (corefile) pour un point de contrôle , avec zéro pour aucune sauvegarde.
G95_CHECKPOINT_MSG	Logique	Si vrai (TRUE), affiche un message sur la sortie standard si le processus a des points de contrôle. Défaut vrai (TRUE).
G95_FPU_ROUND	Chaîne	Définit le mode d'arrondi en virgule flottante. Les valeurs peuvent être vers le plus proche (NEAREST), vers plus l'infini (UP), vers moins l'infini (DOWN), vers zéro (ZERO). Le défaut est NEAREST.
G95_FPU_PRECISION	Chaîne	Précision de résultats intermédiaires. la valeur peut être 24, 53 et 64. Par défaut, c'est 64. Disponible seulement sur x86 et compatibles.

G95_FPU_DENORMAL	Logique	Lève une exception de virgule flottante quand on rencontre un nombre dénormal. Faux par défaut.
G95_FPU_INVALID	Logique	Lève une exception de virgule flottante après une opération invalide. Faux par défaut.
G95_FPU_ZERODIV	Logique	Lève une exception de virgule flottante après une division par zéro. Faux par défaut.
G95_FPU_OVERFLOW	Logique	Lève une exception de virgule flottante après un débordement. Faux par défaut.
G95_FPU_UNDERFLOW	Logique	Lève une exception de virgule flottante après une valeur trop petite (underflow). Faux par défaut.
G95_FPU_INEXACT	Logique	Lève une exception de virgule flottante après une perte de précision.
G95_FPU_EXCEPTIONS	Logique	Précise si les exception de virgule flottante masquées doivent être affichées après l'arrêt du programme. Faux par défaut.
G95_UNIT_x	Chaîne	Remplace le nom par défaut de l'unité x. Le défaut est `fort.`x
G95_UNBUFFERED_x	Logique	Si vrai (TRUE), l'unité x n'est pas tamponnée. Faux par défaut.

Les codes d'erreur d'exécution

Lancer un programme compilé avec g95 en ajoutant l'option `--g95` va afficher cette liste de codes d'erreur sur la sortie standard.

-2	Fin d'enregistrement
-1	Fin de fichier
0	Retour avec succès
	Codes d'erreurs du système d'exploitation (1 - 199)
200	Options incompatibles
201	Option invalide
202	Option manquante
203	Fichier déjà ouvert dans une autre unité
204	Unité non rattachée
205	Erreur de FORMAT
206	ACTION incorrecte spécifiée
207	Tentative de lecture après l'enregistrement ENDFILE (fin de fichier)
208	Valeur incorrecte lors de la lecture
209	Débordement numérique lors de la lecture
210	Pas assez de mémoire
211	Tableau déjà alloué
212	Tentative de désallouer un mauvais pointeur
214	Enregistrement corrompu dans un fichier non formatté en accès séquentiel
215	Tentative de lire plus de données que la taille de l'enregistrement (RECL)
216	Tentative d'écrire plus de données que la taille de l'enregistrement (RECL)

Caractéristiques issues du Fortran 2003

G95 implemente quelques caractéristiques du Fortran 2003. Pour une discussion de toutes les nouvelles caractéristiques du Fortran 2003, voir :
`http://www.kcl.ac.uk/kis/support/cit/fortran/john_reid_new_2003.pdf`.

- Les procédures intrinsèques suivantes sont disponibles: `COMMAND_ARGUMENT_COUNT()`, `GET_COMMAND_ARGUMENT()`, `GET_COMMAND()` et `GET_ENVIRONMENT_VARIABLE()`
- Les variables réelles ou double precision pour les compteurs de boucle `DO` ne sont pas implémentées en g95.
- Le crochets `[` et `]` peuvent être utilisés à la place de `(/` et `/)` pour les constructeurs de tableaux.
- TR 15581 - Types dérivés dynamiques. Permet l'utilisation de l'attribut `ALLOCATABLE` avec des arguments formels, des résultats de fonction et des composants de structures.

- E/S de type flot - L'accès de type flot (stream) du F2003 permet à un programme Fortran de lire et d'écrire des fichiers binaires sans se préoccuper de la structure des enregistrements. Clive Page a écrit de la documentation sur ce sujet, disponible à :

`http://www.star.le.ac.uk/~cgp/streamIO.html`.

- L'instruction `IMPORT`. Utilisé dans le corps d'une interface pour permettre l'accès à un objet de l'unité de programme hôte.
- Convention européenne pour les nombres réels– une étiquette `DECIMAL='COMMA'` dans une instruction `OPEN`, `READ` ou `WRITE` permet le remplacement du point décimal dans les nombres réels par une virgule.
- `MIN()` et `MAX()` fonctionnent avec les caractères aussi bien qu'avec les types numériques.
- Un attribut `VALUE` dans une déclaration de type d'un argument formel fait que l'argument effectif est passé par valeur.
- Les constructeurs de structures style F2003 sont supportés.
- Les pointeurs de procédures style F2003 sont supportés.
- La fonction incorporée `move_alloc()` est supporté.
- Le support IEEE du F2003 est moitié implémenté.
- La construction `BIND(C)` du F2003 et le module `ISO_C_BINDING` qui fournissent une meilleur interopérabilité avec le C.

S'interfacer avec les programmes g95

Bien que g95 produise des exécutables autonomes, il est parfois souhaitable de s'interfacer avec d'autres programmes, souvent en C. La première difficulté que recontrera un programme multi-langage est le nom des symboles publics. G95 suit la convention f2c d'ajouter un caractère souligné aux noms publics, ou deux si le nom contient déjà un caractère souligné. Les options `-fno-second-underscore` et `-fno-underscoring` peuvent être utiles pour forcer g95 à fournir des noms compatibles avec votre compilateur C. Utilisez le programme `nm` pour examiner les fichiers `.o` produits par les deux compilateurs. G95 met aussi les noms publics en minuscules, à moins que `-fupper-case` soit précisé, auquel cas tout sera en majuscules. Les noms de modules sont représenté par *nom-module*`_MP_`*nom-entité*.

Après l'édition de liens, il y a deux cas: Le Fortran appelle des procédures en C et le C appelle des procédures en Fortran. Pour le C qui appelle des procédures en Fortran, les procédures Fortran vont souvent appeler les procédures de la bibliothèque Fortran qui s'attend à ce que le tas soit initialisé d'une certaine façon. Pour forcer une initialistion manuelle à partir du C, appeler `g95_runtime_start()` pour initialiser la bibliothèque Fortran et `g95_runtime_stop()` quand tout est fini. Le prototype de `g95_runtime_start()` est :

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

La bibliothèque doit être capable de traiter les options de la ligne de commande. Si c'est difficile à faire et que votre programme n'a pas besoin d'arguments de ligne de commande, passez `argc=0` et `argv=NULL`. Sous OSX, ajoutez `-lSystemStubs` quand vous utilisez g95 pour lancer l'éditeur de liens et lier des fichier objets compilés avec GCC.

F2003 fournit un certain nombre de fonctionnalités qui rendent plus facile l'intrefaçage avec le C. L'attribut `BIND(C)` permet de créer des symboles Fortran qui sont plus facile à référencer à partir du C (ou d'un autre langage). Par exemple:

```
SUBROUTINE toto(a) BIND(C)
```

Cette ligne crée un symbole `toto` sans altération du nom avec des caractères souligné. Tous les caractères sont mis en minuscules. Une forme similaire est :

```
SUBROUTINE toto(a) BIND(C, name='Toto1')
```

Ceci fait que le nom du symbole est `Toto1`. Au niveau du Fortran, la procédure est toujours référencée par le `toto` habituel, ou `TOTO` ou n'importe quelle autre combinaison.

Les programmes en C passsent leurs arguments par valeur, alors que le Fortran les passe par référence. F2003 fournit l'attribut `VALUE` pour signaler des arguments formels qui sont passés par valeur. Un exemple serait :

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

Une procédure définie de cette façon peut toujours être appellée à partir du Fortran avec la restriction que les arguments formels ne sont plus directement associés aux arguments effectifs et donc que modifier un argument formel ne modifie plus l'argument effectif.

On peut accéder aux variables globales de façon similaire. La procédure suivante affiche la valeur de la variable `VAR`, qui autrement serait inaccessible au Fortran :

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

Quand le Fortran considère que chaque type a différentes variantes, le C définit tout comme des types distincts. Afin de désigner le même objet, le F2003 fournit un module intrisèque, `ISO_C_BINDING` qui contient les correspondances entre les variantes du Fortran et les types du C. Quand il est utilisé avec `USE`, les constantes `PARAMETER` suivantes sont définies :

`c_int`	Variante entière pour les `int` du C
`c_short`	Variante entière pour les `short` du C
`c_long`	Variante entière pour les `long` du C
`c_long_long`	Variante entière pour les `long long` du C
`c_signed_char`	Variante entière pour les `char` du C
`c_size_t`	Variante entière pour les `size_t` du C
`c_intptr_t`	Entier de la même taille qu'un pointeur C
`c_float`	Variante réelle pour les `float` du C
`c_double`	Variante réelle pour les `double` du C

Il y a bien d'autre choses dans `ISO_C_BINDING`. En utilisant ce module, on peut écrire un programme de ce genre :

```
SUBROUTINE toto
    USE, INTRINSIC :: ISO_C_BINDING
    INTEGER(KIND=C_INT) :: entier_var
    INTEGER(KIND=C_LONG_LONG) :: gros_integer
    REAL(KIND=C_FLOAT) :: reel_var
    ...
```

Utiliser le générateur de nombres aléatoires

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

Retourne dans `recolte` les nombres aléatoires sous la forme d'un scalaire `REAL` ou un tableau de `REAL`, $0 \leq$ `recolte` < 1.

Changer la graine du générateur aléatoire:

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

`sz` est le nombre minimum d'entiers du type pris par défaut nécessaires pour contenir la valeur de la graine; g95 retourne 4. L'argument `pt` est un tableau d'entiers du type pris par défaut de taille `n1` $\geq$ `sz`, contenant les valeurs de la graine fournie par l'utilisateur. L'argument `gt` est un tableau d'entiers du type pris par défaut de taille `n2` $\geq$ `sz`, contenant la graine actuelle.

Appeler `RANDOM_SEED()` sans arguments initialise la graine à une valeur déterminée par la valeur de l'horloge. Ceci peut être utiisé pour générer des séquences pseudo-aléatoires qui sont différentes pour chaque appel du programme. La graine sera aussi initialisée avec une valeur dépendant du temps au démarage du

programme si la variable d'environnement `G95_SEED_RNG` est mise à TRUE. Si aucune de ces conditions n'est vérifiée, `RANDOM_NUMBER()` génèrera toujours la même séquence.

Le générateur utilisé est le générateur à décalage et ou exclusif développé par George Marsaglia.

Macros Prédéfinies du Préprocesseur

Les macros qui sont toujours définies sont :

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

Les macros conditionnelles sont:

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

La reprise à partir d'un fichier image

Sur les systèmes x86 linux, l'exécution d'un programme compilé par g95 peut être suspendue et reprise plus tard. Si vous suspendez votre programme ne lui envoyant le signal QUIT, qui est souvent obtenu avec contrôle-backslash, le programme écrira un fichier exécutable nommé `dump` dans le répertoire courant. Lancer cet exécutable provoque la reprise de l'exécution dee votre programme à partir du point où le fichier `dump` à été généré. La session suivante le montre:

```
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      (contrôle-\ tap)
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 %
```

Tous les fichiers ouverts doivent présents au même endroit que dans le processus original. Si vous effectuez des liaisons avec un autre langage, ceci peut ne pas fonctionner. Bien que l'utilisation principale est de conserver l'état d'une exécution en cas de plantage et de redémarrage de la machine, d'autres possibilités incluent de mettre une très longue tâche dans une file d'attente ou de déplacer une tâche d'un ordinateur à un autre. On peut effectuer des points de contrôles automatiques de votre programme en mettant dans la variabe d'environnement `G95_CHECKPOINT` le nombre de secondes à attendre entre deux sauvegardes. Une valeur de zéro signifie pas de sauvegarde. Les nouveaux fichiers de point de contrôle écrasent les anciens.

Compilation intelligente

Considérons un module `toto` dont le code source est dans le fichier `toto.f95`. On peut distinguer deux types de modifications de `toto.f95`:

1. Les modifications qui changent l'utilisation de `toto`, par exemple en changeant l'interface d'une procédure;
2. Les modifications qui ne changent pas l'utilisation de `toto`, mais seulement son implémentation, par exemple en corrigeant un bug dans le corps d'une procédure.

Les deux types de modifications vont en général changer le fichier objet `toto.o`, mais seules celles du premier type vont changer le contenu de `toto.mod`. Quand il recompile un module, g95 est suffisament malin pour détecter si le fichier `.mod` a besoin d'être mis à jour : après une modification de type 2, l'ancien fichier `.mod` est conservé.

Cette fonctionnalité de g95 évite des compilations en cascade inutiles quand on reconstruit un gros programme. En effet, supposons que beaucoup de fichiers sources différents dépendent de `toto.mod`, soit directement (à cause d'une instruction `USE toto`) soit indirectement (en utilisant un module qui utilise `toto`, etc...). Une modification de `toto.f95` de type 1va déclencher une recompilation de tous les fichiers sources dépendants; heureusement de telles modificarions ont de fortes chances d'être assez rares. Les modifications de type 2, plus courantes vont seulement provoquer une recompilation de `toto.f95` lui-même, après laquelle le nouveau fichier objet `toto.o` peut être immédiatement lié aux autres fichiers objets existants pour créer un programme exécutable mis à jour.

Les fonctions intrinsèques supplémentaires de g95

ACCESS

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

Vérifie si on peut accéder au fichier filename avec le mode spécifié, où mode est une ou plusieurs des lettres `rwxRWX`. Retourne zéro si les permissions sont accordées, un nombre différent de zéro si quelquechose ne va pas.

ALGAMA

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

Retourne le logarithme naturel de $\Gamma(x)$. `ALGAMA` est une fonction générique qui prend n'importe quelle variante de réel.

BESJ0

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

Retourne la fonction de Bessel d'ordre zéro de première espèce. Cette fonction est générique et prend n'importe quelle variante de réel.

BESJ1

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

Retourne la fonction de Bessel d'ordre un de première espèce. Cette fonction est générique et prend n'importe quelle variante de réel.

BESJN

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

Retourne la fonction de Bessel d'ordre n de première espèce. Cette fonction est générique et prend n'importe quelle variante de réel.

BESY0

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

Retourne la fonction de Bessel d'ordre zéro de seconde espèce. Cette fonction est générique et prend n'importe quelle variante de réel.

BESY1

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

Retourne la fonction de Bessel d'ordre un de seconde espèce. Cette fonction est générique et prend n'importe quelle variante de réel.

BESYN

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

Retourne la fonction de Bessel d'ordre n de seconde espèce. Cette fonction est générique et prend n'importe quelle variante de réel.

CHMOD

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

Change les permissions unix pour un fichier. Retourne une valeur non nulle en cas d'erreur.

DBESJ0

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

Retourne la fonction de Bessel d'ordre zéro de première espèce.

DBESJ1

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

Retourne la fonction de Bessel d'ordre un de première espèce.

DBESJN

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

Retourne la fonction de Bessel d'ordre n de première espèce.

DBESY0

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

Retourne la fonction de Bessel d'ordre zéro de seconde espèce.

DBESY1

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

Retourne la fonction de Bessel d'ordre un de seconde espèce.

DBESYN

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

Retourne la fonction de Bessel d'ordre n de seconde espèce.

DCMPLX

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

Version double precision de `CMPLX`, `x` et `y` peuvent être de n'importe quel type ou variante numérique.

DERF

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

Retourne la fonction erreur de `x` en double précision.

DERFC

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

Retourne la fonction erreur complémentaire de `x` en double précision.

DFLOAT

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

Convertit un nombre `x` en double précision. C'est un synonyme de la fonction intrinsèque `DBLE`.

DGAMMA

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

Retourne $\Gamma(x)$ en double précision.

DLGAMA

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

Retourne le logarithme naturel de $\Gamma(x)$.

DREAL

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

Convertit un nombre `x` en double précision. C'est un synonyme de la fonction intrinsèque `DBLE`.

DTIME

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

Met dans `tarray(1)` le nombre de secondes de temps utilisateur écoulées dans le processus depuis la dernière fois que `DTIME` a été appellée. Met dans `tarray(2)` le nombre de secondes de temps système écoulées dans le processus depuis la dernière fois que `DTIME` a été appellée. Retourne la somme de ces deux temps.

ERF

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

Retourne la fonction erreur de `x`. Cette fonction est générique et prend n'importe quelle variante de réel.

ERFC

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

Retourne la fonction erreur complémentaire de `x`. Cette fonction est générique et prend n'importe quelle variante de réel.

ETIME

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

Met dans `tarray(1)` le nombre de secondes de temps utilisateur écoulées dans le processus courant. Met dans `tarray(2)` le nombre de secondes de temps système écoulées dans le processus courant. Retourne la somme de ces deux temps.

FNUM

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

Retourne le numéro descripteur de fichier correspondant à l'unité `unit`. Retourne -1 si l'unité n'est pas connectée.

FSTAT

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

Obtient des informations sur un fichier ouvert à partir de l'unité d'E/S Fortran `unit` et les place dans le tableau `sarray()`. Les valeurs de ce tableau sont extrites de la structure stat qui est retournée par `fstat(2)`, comme suit:

`sarray(1)` Numéro de périphérique,
`sarray(2)` Numéro d'inode,
`sarray(3)` mode fichier,
`sarray(4)` nombre de liens,
`sarray(5)` UID du propriétaire,
`sarray(6)` GID du groupe,
`sarray(7)` Type de périphérique,
`sarray(8)` Taille du fichier,

`sarray(9)` Dernier accès,
`sarray(10)` Dernière modification,
`sarray(11)` Temps modification,
`sarray(12)` Taille des blocs,
`sarray(13)` Blocs allouées.

FDATE

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

Retourne l'heure et la date actuelles sous la forme : Day Mon dd hh:mm:ss yyyy.

FTELL

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

Retourne la position actuelle (offset) de l'unité `unit` ou -1 si `unit` n'est pas ouvert.

GAMMA

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

Retourne une approximation de $\Gamma(x)$. `GAMMA` est une fonction générique qui prend toute variante de réel.

GETCWD

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

Retourne le nom du répertoire de travail actuel dans `name`. Retourne une valeur non nulle en cas d'erreur.

GETGID

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

Retourne l'identificateur de groupe (GID) du processus courant.

GETPID

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

Retourne l'identificateur de processus (PID) du processus courant.

GETUID

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

Retourne l'indentificateur de l'utilisateur (UID).

HOSTNM

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

Met dans `name` le nom du système hôte (host name). Retourne une valeur non nulle en cas d'erreur.

IARGC

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

Retourne le nombre d'arguments de la ligne de commande (sans compter le nom du programme lui-même).

ISATTY

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

Retourne `.true.` si et seulement si l'unité d'E/S Fortran spécifiée par`unit` est connectée à un terminal.

ISNAN

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

Retourne `.true.` si `x` est un Not-a-Number (NaN). Cette fonction est générique.

LINK

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

Crée un lien (dur ou *hard link*) de `path1` jusqu'à `path2`.

LNBLNK

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

Synonyme pour la fonction intrisèque `len_trim`. Retourne la position du dernier caractère non blanc dans la chaîne string.

LSTAT

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

Si `file` est un lien symbolique, retourne des informations sur le lien lui-même. Voir la fonction `FSTAT()` pour plus de détails. Retourne une valeur non nulle en cas d'erreur.

RAND

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

Retourne un nombre pseudo-aléatoire dstribué uniformément, tel que $0 \leq$ `rand` < 1. Si `x` est 0, le nombre suivant de la séquence est retourné. Si `x` est 1, le générateur est réinitialisé en appelant `srand(0)`. Si `x` a n'importe quelle autre valeur, elle est utilisée comme graine pour `srand`.

SECNDS

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

Retourne le temps local en secondes depuis minuit moins la valeur `t`. Cette fonction est générique.

SIGNAL

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

Interface avec l'appel système `signal` d'Unix. Retourne une valeur non nulle en cas d'erreur.

SIZEOF

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

L'argument `object` est le nom d'une expression ou un type. Retourne la taille de l'objet `object` en octets.

STAT

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

Obtient des information au sujet du fichier `file` et les place dans le tableau `sarray`. Voir la fonction `fstat()` pour plus de détails. Retourne une valeur non nulle en cas d'erreur.

SYSTEM

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

Appelle la commande externe placée dans la chaîne `cmd`. Retourne le code de sortie système.

TIME

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

Retourne le temps courant codé sous la forme d'un entier à la manière de la fonction Unix `time`.

UNLINK

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

Défait le lien vers le fichier`file` (le détruit). Retourne une valeur non nulle en cas d'erreur.

%VAL()
Quand elle est appliquée à une variable dans la liste des paramètres formels, elle fait que la variable est passée par valeur. Cette pseudo-fonction n'est pas recommendée, et n'est implémentée que par souci de compatibilité. L'attribut `VALUE` du F2003 est le mécanisme normal pour faire celà.

%REF()
Quand elle est appliquée à une variable dans la liste des paramètres formels, elle fait que la variable est passée par référence.

Procédures intrinsèques supplémentaires de g95

ABORT

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

Force le programme à s'arrêter avec génération d'un fichier core, en s'envoyant un singal SIGABORT à lui-même 5Unix).

CHDIR

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

Fait de `dir` le répertoire de travail courant.

DTIME

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

Met dans `tarray(1)` le nombre de secondes de temps utilisateur écoulées depuis le dernier appel à `DTIME`. Met dans `tarray(2)` le nombre de secondes de temps système écoulées depuis le dernier appel à `DTIME`. Met dans `result` la somme de ces deux temps.

ETIME

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

Met dans `tarray(1)` le nombre de secondes de temps utilisateur écoulées dans le processus courant. Met dans `tarray(2)` le nombre de secondes de temps système écoulées dans le processus courant. Met dans `result` la somme de ces deux temps.

EXIT

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

Arrête un programme avec le statut `code` après avoir fermé les unités d'E/S Fortran restées ouvertes. Cette procédure est générique.

FDATE

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

Met dans `date` l'heure et la date courante au format: Day Mon dd hh:mm:ss yyyy.

FLUSH

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

Purge (flush) L'unité Fortran `unit` actuellement ouverte en écriture.

FSTAT

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

Obtient des informations sur le fichier ouvert sur l'unité d'E/S Fortran `unit` et les place dans le tableau `sarray()`. Met `status` à une valeur non nulle en cas d'erreur. Voir la fonction `fstat` pour avois des informations sur la façon dont `sarray` est organisé.

GETARG

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

Met dans `value` le paramètre de ligne de commande qui se trouve en position `pos`.

GETENV

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

Récupère la variable d'environnement `variable` et met sa valeur dans `value`.

GETLOG

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

Retourne le nom de login pour le processus dans `name`.

IDATE

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

Met dans `m` le mois courant, dans `d` le jour courant et dans `y` l'année courante. Cette procédure n'est pas très protable d'une implémentation à l'autre. Utilisez la procédure standard `DATE_AND_TIME` dans tout nouveau code.

ITIME

```
SUBROUTINE itime(tarray)
    INTEGER, INTENT(OUT) :: tarray(3)
END SUBROUTINE itime
```

Retourne le temps local, heure, minute et seconde, respectivement dans les éléments 1,2 et 3 de `tarray`

LSTAT

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

Si `file` est un lien symbolique, retourne des informations sur le lien lui-même. Voir la fonction `FSTAT()` pour plus de détails.

LTIME

```
SUBROUTINE ltime(stime,tarray)
    INTEGER :: stime
    INTEGER, INTENT(OUT) :: tarray(9)
END SUBROUTINE ltime
```

Convertit le temps système `stime` en temps local, donné dans `tarray`, un tableau d'entiers à 9 éléments. Les éléments de `tarray` sont :

`tarray(1)` Secondes
`tarray(2)` Minutes
`tarray(3)` Heures après minuit
`tarray(4)` Nombre de mois depuis Janvier
`tarray(5)` Jour du mois
`tarray(6)` Année depuis 1900
`tarray(7)` Nombre de jours depuis Dimanche
`tarray(8)` Jours depuis le 1er Janvier
`tarray(9)` Indicateur de l'heure d'été : positif, l'heure d'été est activée, zéro sinon et négatif si l'information n'est pas disponible.

RENAME

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

Change le nom du fichier de `path1` en `path2`. Si l'argument `status` est fourni, il est ms à une valeur non nulle en cas d'erreur.

SECOND

```
SUBROUTINE second(time)
    REAL, INTENT(OUT) :: time)
END SUBROUTINE second
```

Retourne le temps d'excution `time` du processus en seconde.

SIGNAL

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

Interface avec l'appel système `signal` d'Unix. Met dans `status` une valeur non nulle en cas d'erreur.

SLEEP

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

Force le processus à effectuer une pause de `seconds` secondes.

SRAND

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

Réinitialise le générateur aléatoire. Voir la fonction `srand()` pour plus de détails.

STAT

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

Obtient des informations sur le fichier `file` et les place dans le tableau `sarray`. Voir `fstat()` pour plus de détails. Met une valeur non nulle dans `status` en cas d'erreur.

SYSTEM

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

Passe la commande `cmd` à un interpréteur. Si `result` est précisé, il prendra la valeur du code de sortie système de `cmd`.

UNLINK

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

Défait le lien vers (c'est-à-dire détruit) le fichier `file`. En cas d'erreur `status` prend une valeur non nulle