G95

It's Free Crunch Time
http://www.g95.org

Основные характеристики G95

- Бесплатный компилятор языка Фортран 95.
- Текущая (на Сентябрь 2006) версия g95 - 0.91.
- Лицензия - Открытое Лицензионное Соглашение (GNU Open Source, GPL license).
- Выполнение скомпилированных программ может быть изменено с помощью большого количества переменных окружения, документированных в самой скомпилированной программе.
- TR15581 – Динамически размещаемые формальные параметры и компоненты производных типов.
- Указатели на процедуры, структуры, встроенная совместимость с СИ - Фортран 2003
- Встроенные процедуры и модули из Фортрана 2003.
- Передача формальных параметров типа `VALUE` в подпрограмму по значению.
- Использование запятой как разделителя в десятичных дробях в операторах OPEN, READ и WRITE.
- Квадратные скобки `[` и `]` могут быть использованы в конструкторах массивов.
- Оператор `IMPORT` используется в теле интерфейса для доступа к объектам объемлющей контекстной области.
- `MIN()` и `MAX()` для символьных данных, как и для чисел.
- `OPEN` для "Transparent" или потокового Ввода/Вывода (В/В).
- Обратная бинарная совместимость с g77 (ABI).
- Доступны целочисленные типы данных по умолчанию размером 32 и 64 бит.
- Исполнение системных команд с помощью `SYSTEM()`.
- Возможность использования символа табуляции в исходном коде.
- Имена переменных могут содержать $.
- Холлеритовы константы.
- DOUBLE COMPLEX расширение.
- Переменная длина для именованных `COMMON` блоков.
- Смешение числовых и символьных типов в `COMMON` и `EQUIVALENCE`.
- `INTEGER` целочисленные типы: 1, 2, 4, 8.
- `LOGICAL` логические типы: 1, 2, 4, 8.
- `REAL` вещественные типы: 4, 8.
- `REAL(KIND=10)` вещественные типы для x86-совместимых систем. Точность до 19 знака, значения в пределах $10^{\pm 4931}$.
- Вывод вещественных данных, управляемый списком, печатает минимальное количество знаков после запятой, достаточное для адекватного представления числа.
- Отладочные строки, начинающиеся с (D) из VAX.
- Операции с символьными константами с использованием символов из Си (например 'hello\nworld').
- Дескрипторы \ и $.
- Встроенные системные функции из VAX (SECNDS и т.д.)
- Системные функции из Unix (getenv, etime, stat и т.д.)
- Выявление неразмещенных и неконформных (выходящих за заявленные границы) массивов во время исполнения - см. Таблицу IV:
  `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`
- Выявление утечек памяти - см. Таблицу V:
  `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`
- Указание места ошибки исполнения в исходном коде.
- "Умная" компиляция предотвращает каскадную перекомпиляцию модулей.
- Возможность задания инструкции совместимости с языком F. См. `http://www.fortran.com/F`. G95 можно собрать в качестве компилятора языка F.
- Приостановка и возобновление исполнения с места остановки на x86/Linux.
- Устаревшая возможность использования вещественных переменных цикла УДАЛЕНА.
- Как правило, быстрый отклик разработчика на сообщения об ошибках.
- Возможность сборки с версиями GCC 4.0.3 и 4.1.1.
- Работает на Linux/x86, PowerPC, 64-bit Opteron, 64-bit Itanium, 64-bit Alpha.
- Работает на Windows/Cygwin, MinGW, и Interix.
- Работает на OSX на Power Mac G4, x86-OSX.
- Работает на FreeBSD на x86, HP-UX 11, Sparc-Solaris, x86-Solaris, OpenBSD, NetBSD, AIX, IRIX, Tru64 UNIX на процессорах Alpha.
- Так же доступны fink-версии (для макинтошей).
- Собранные бинарные стабильные и текущие версии для большинства платформ доступны на `http://ftp.g95.org`.

Периодически я встречаюсь с людьми, с которыми переписывался по поводу g95. Часто они говорят мне, какой великолепный продукт я создаю в одиночку. Я всегда смеюсь и указываю на то, что я никогда не работал над этим в одиночку. Число людей, которые активно помогали, и помогают, в создании g95, возможно, около тысячи. Считается, что человек, который пишет код, делает всю работу, в то время, как люди, которые отлавливают и выделяют ошибки до нескольких строк кода в действительности выполняют огромную работу – работу, которую обычно не оценивают по достоинству. Написать что-либо схожее по сложности с современным компилятором фортрана невозможно в одиночку. Уж я-то знаю.

Как множество других вещей, g95 родился как результат чувства неудовлетворенности. Я писал код для своей диссертации на Фортране 77, используя g77. Фортран - это замечальный язык для численных расчетов – он очень удобен для людей, которые больше озабочены получением ответа, чем программированием. Моя программа содержала множество достаточно сложных структур – связанные списки, октарные деревья, разреженные матрицы, генерацию сетки на конечных элементах, решение уравнения Пуассона, многополюсные разложения, минимизация методом сопряженных градиентов и множество геометрических расчетов. Из-за использования Фортрана 77, код получился “некрасивым” и мог бы быть намного лучше, если бы использовал динамические массивы и производные типы данных. Да и моя диссертация подходила к концу, и мне нужна была новая цель.

Кроме удобства более продвинутых особенностей языка, меня также сильно вдохновляли работы Билла Каэна (Bill Kahan). Вывод, к которому я пришел, прочитав его статьи, был таков - хотя численные расчеты очень коварны, но есть способы минимизировать ошибки до такой степени, что они уже не будут иметь значения. И тут пользователь часто оставлен на милость создателей библиотек.

Компилятор – это прикольно, но как раз библиотеки интересуют меня больше. Поведение компилятора достаточно жестко определяется стандартом языка, но именно в библиотеках проявляются новшества. Даже на примитивном уровне, в библиотеке больше всяких фишек в сравнении с другими продуктами. Возможность возобновления исполнения – это такая штука, которую я хотел задолго до того, как появилась возможность это реализовать.

Работа над g95 очень интересна, и я надеюсь продолжать ее в течение еще долгого времени.

Энди Вот (Andy Vaught)
Меса, Аризона (Mesa, Arizona)
Октябрь 2006

Примечание “переводчика”.

Главной целью данного перевода является попытка приобщить большее число людей к программированию на Фортране, в частности, с использованием компилятора G95. Английский язык, конечно же, необходим каждому программисту (в обычном смысле этого слова), но Фортран – это язык научных расчетов, одной из целей создателей которого было предоставление возможности неспециалистам пользоваться компьютерами для расчетных задач. Но так уж повелось, что английским языком на достаточном уровне владеют все же немногие люди, и даже те, кто думает, что владеют, на самом деле часто заблуждаются – говорю по собственному опыту.

Я не являюсь профессиональным переводчиком ни в коей мере. Более того, большую часть того, что я знаю о программировании, я изучал в англоязычной обстановке, и поэтому частенько русскоязычные аналоги, казалось бы, тривиальных терминов на английском в моей голове просто отсутствуют (также они отсутствуют и в доступных мне словарях). Поэтому прошу не судить строго данную работу, но конструктивная критика и советы приветствуются.

Еще одной из причин, по которой я решил этим заняться, была возможность внести хоть какой-то вклад в движение программ с открытыми исходниками. Нет, у меня нет каких-то религиозных чувств по отношению к открытому коду, но, как говорится “give and take”, так что, чем могу, тем и помогаю в благодарность за возможность использования множества других программ.

В общем, если нравится, то пользуйтесь на здоровье. Если нет, то значит, просто не судьба. Если есть вопросы, то пишите на victor.prosolin@gmail.com.

Виктор Просолин
Калгари, Альберта, Канада
Январь 2007

Лицензия на использование

G95 лицензирован по Открытому лицензионному соглашению GNU (GNU General Public License (GPL)). Все юридические нюансы можно посмотреть тут `http://www.gnu.org/licenses/gpl.html`.

Подключаемая во время выполнения библиотека лицензирована в основном под GPL и содержит одно исключение, которое дает пользователям право использовать библиотеки, скомпилированные g95, в программах, не подчиняющимися GPL, без обязательного лицензирования конечного продукта под GPL или любого другого влияния GPL на него.

Установка

Unix (Linux/OSX/Solaris/Irix/etc.):
Откройте консоль, перейдите в директорию, в которую вы планируете установить g95. Чтобы скачать и установить g95, исполните следующие команды:

```
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
```

Следующие файлы и директории должны появиться в результате установки:

```
./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
```

Файл `cc1` является символьной ссылкой на `f951` в той же директории.

Cygwin:

Инструкция `-mno-cygwin` позволяет Cygwin-версии g95 создавать файлы, которые не требуют библиотеки cygwin1.dll, и таким образом, могут исполняться на других машинах. Также такие исполняемые файлы избавлены от ограничений GPL. Чтобы установить Cygwin-версию с работающей -mno-cygwin инструкцией, нужно установить библиотеки mingw, которые можно взять на сайте проекта Cygwin `http://www.cygwin.com`. Скачайте архив с `http://ftp.g95.org/g95-x86-cygwin.tgz` в корневую директорию Cygwin (обычно c:\Cygwin). Запустите Cygwin и выполните следующие команды:

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

Это установит g95 в папку `/usr/local/bin`. Осторожно: Не используйте Winzip для распаковки архива, иначе некоторые необходимые ссылки могут установиться неправильно.

MinGW:

Бинарные файлы g95 для MS-Windows находятся в самораспаковывающемся инсталляторе. На данный момент доступны 2 версии. Пользователи Windows 98 должны использовать g95, собранный с использованием gcc 4.0.3, `http://ftp.g95.org/g95-MinGW.exe`. Пользователи Windows NT, XP и 2000 могут использовать тот же пакет или выбрать сборку с gcc 4.1.1, доступную на `http://ftp.g95.org/g95-MinGW-41.exe`.

Бесплатный пакет MinGW/Msys предоставляет файлы из GNU GCC, которые необходимы для работы g95, среди которых линкер `ld.exe` и GNU ассемблер `as.exe` из binutils. Его можно скачать тут `http://www.mingw.org`. Версия g95 для MinGW предоставляет выбор из 2х возможных типов установки. Если MinGW не найден, то g95 вместе с необходимыми программами и библиотеками binutils установится в выбранную пользователем директорию. Добавьте эту директорию в переменную `PATH` и в `LIBRARY_PATH`.

Если MinGW установлен, то рекомендуется установить g95 в корневую директорию MinGW (обычно `C:\mingw`) во избежание возможных конфликтов. Если инсталлятор найдет MinGW, то он попытается установить g95 в файловую систему MinGW. Добавьте директорию `MinGW\bin` в переменную `PATH` и задайте `LIBRARY_PATH=`*путь-к-MinGW*`/lib`

На Windows 98 и Me процесс установки требует редактирования системного файла `autoexec.bat` и перезагрузки.

На заметку пользователям Windows XP: MinGW в настоящее время выделяет только 8 мегабайт в динамической памяти (heap). Если вашей программе требуется больше памяти, попробуйте скомпилировать с `-Wl,-heap=0x01000000`. Подберите шестнадцатиричное значение для `-heap`, достаточное для того, чтобы ваша программа работала.

Запуск G95

G95 определяет тип компиляции в зависимости от расширения файла. Допустимые расширения файлов Фортрана ограничены следующими – `.f`, `.F`, `.for`, `.FOR`, `.f90`, `.F90`, `.f95`, `.F95`, `.f03` and `.F03`. В каком формате (то есть в фиксированном или свободном) будет обрабатываться исходный файл, определяется его расширением. Расширения `.f`, `.F`, `.for` и `.FOR` считаются написанными в фиксированном формате, совместимым со старым f77. Расширения `.f90`, `.F90`, `.f95`, `.F95`, `.f03` и `.F03` считаются написанными в свободном формате. Расширения, начинающиеся с заглавной буквы F, будут обработаны Си-препроцессором по умолчанию, а начинающиеся с маленькой f по умолчанию им не обрабатываются.

Основные инструкции для компилирования Фортрановских файлов:

`-c` Компилировать, но не запускать линкер.

`-v` Показать программы, вызываемые g95 и их аргументы. В частности, полезно для отслеживания проблем с путями.

`-o` Задать имя выходного – объектного или исполняемого – файла. Расширение `.exe` автоматически добавляется в Windows. Если выходной файл не задан, то в Unix по умолчанию создается `a.out`, а в Windows `a.exe`

Примеры:

`g95 -c hello.f90`
Компилирует `hello.f90` в объектный файл с именем `hello.o`.

`g95 hello.f90`
Компилирует `hello.f90` и линкует его в исполняемый файл `a.out` (в unix) или `a.exe` (в MS Windows).

`g95 -c h1.f90 h2.f90 h3.f90`
Компилирует несколько исходных файлов. Если ошибок не найдено, то создаются объектные файлы `h1.o`, `h2.o` и `h3.o`.

`g95 -o hello h1.f90 h2.f90 h3.f90`
Компилирует несколько исходных файлов, линкует и создает `hello` в unix, или `hello.exe` в MS Windows .

Краткий обзор инструкций компилятора

`g95 [ -c | -S | -E ]`	Компилировать & собрать | Создать код ассемблера |Показать исходный код
`[-g] [-pg]`	Инструкции для отладки
`[-O[`*n*`] ]`	Уровень оптимизации, $n = 0, 1, 2, 3$
`[-s ]`	Убрать отладочную информацию
`[-W`*warn* `] [-pedantic]`	Инструкции предупреждений
`[-I`*dir* `]`	Добавить директорию для поиска включаемых (#include) файлов
`[-L`*dir* `]`	Добавить директорию для поиска библиотек
`[-D macro[=value]... ]`	Определить макрос для препроцессора
`[-U macro ]`	Отменить определение макроса для препроцессора
`[-f option ...]`	Общие инструкции компилятору
`[-m machine-option ...]`	Опции для данной машины. См. GCC документацию
`[-o outfile ]`	Имя выходного файла
`входящий файл`	

Инструкции G95

Использовать: g95 [инструкции] файл(ы)...

`-pass-exit-codes`	Выйти с самым большим кодом ошибки из фазы (компиляции).
`-help`	Показать справочную информацию.
`-target-help`	Показать инструкции для конкретного объекта. (Используйте '`-v -help`' для просмотра опций командной строки для каждого вызываемого подпроцесса).
`-dumpspecs`	Показать все встроенные спецификации.
`-dumpversion`	Показать версию компилятора.

`-dumpmachine`	Показать процессор, для которого собран компилятор.
`-print-search-dirs`	Показать директории, в которых компилятор осуществляет поиск (включаемых файлов, библиотек, исполняемых файлов и т.д.).
`-print-libgcc-file-name`	Показать имя и путь вспомогательной библитеки (libgcc.a).
`-print-file-name=`*библиотека*	Показать полный путь к указанной *библиотеке*.
`-print-prog-name=`*программа*	Показать полный путь к указанному компоненту компилятора *программа*.
`-print-multi-directory`	Показать корневую папку для версий libgcc.
`-print-multi-lib`	Показать способ отображения между опциями командной строки и различными директориями для поиска библиотек.
`-print-multi-os-directory`	Показать относительный путь к системным библиотекам.
`-Wa,`*инструкции*	Передать *инструкции* (через запятую) для ассемблера.
`-Wp,`*инструкции*	Передать *инструкции* (через запятую) для препроцессора.
`-Wl,`*инструкции*	Передать *инструкции* (через запятую) для линкера.
`-Xassembler` *параметр*	Передать *параметр* для ассемблера.
`-Xpreprocessor` *параметр*	Передать *параметр* для препроцессора.
`-Xlinker` *параметр*	Передать *параметр* для линкера.
`-save-temps`	Не удалять временные файлы.
`-pipe`	Использовать конвейеры (pipes) вместо временных файлов.
`-time`	Замерить время исполнения каждого подпроцесса. На некоторых платформах эта функция недоступна (MinGW, OSX).
`-specs=`*файл*	Заменить встроенные спецификации другими из *файл*а.
`-std=`*стандарт*	Считать, что входящий файл соответствует заданному *стандарту*.
`-B` *директория*	Добавить *директорию* для поиска.
`-b` *машина*	Использовать gcc для заданной *машины*, если возможно.
`-V` *версия*	Использовать gcc для заданной *версии*, если таковая имеется.
`-v`	Показать программы, запускаемые g95.
`-M`	Вывести зависимости для Makefile на стандартный вывод.
`-###`	Делает то же, что и `-v`, но ставит опции в кавычки и не выполняет команды.
`-E`	Только обработка препроцессором; без компиляции, создания кода ассемблера или линковки.
`-S`	Только компилировать; не создавать код ассемблера или линковать.
`-c`	Только компилировать и создать код ассемблера, но не линковать.
`-o` *файл*	Задать выходной *файл*.
`-x` *язык*	Задать *язык* входящих файлов. Варианты: c, c++, assembler, none. 'none' означает использовать распознавание по умолчанию, основываясь на расширении.

Инструкции, начинающиеся с `-g`, `-f`, `-m`, `-O`, `-W`, или `-param` автоматически передаются подпроцессам, вызываемым g95. Для того, чтобы передать какие-либо другие инструкции этим подпроцессам, можно использовать `-W`*инструкция*. Инструкции по сообщению об ошибках см. на `http://www.g95.org`.

По умолчанию, программы компилируются без оптимизации. Параметр n в `-O`n задает степень оптимизации, от 0 до 3. 0 (ноль) – значит без оптимизации, и чем выше n, тем выше оптимизация. Задание уровня оптимизации дает компилятору право изменять код, чтобы добиться ускорения исполнения. Результаты часто подвержены влиянию оптимизации. Использование `-O` означает то же, что и `-O1`.

Значительное ускорение может быть достигнуто путем задания следующих инструкций – `-O2 -march=`*arch*, где *arch* – это архитектура используемого процессора, т.е. `pentium4`, `athlon`, `opteron`, и т.д. Еще часто задаваемые инструкции – `-funroll-loops`, `-fomit-frame-pointer`, `-malign-double` и `-msse2`. Информацию о всех опциях GCC см. `http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc`.

Инструкции для препроцессора

G95 работает с файлами, содержащими инструкции Си-препроцессора.

`-cpp`	Безусловно прогнать входящие файлы через препроцессор
`-no-cpp`	Безусловно НЕ прогонять входящие файлы через препроцессор
`-D имя[=значение]`	Определить макрос для препроцессора
`-U имя`	Отменить макрос для препроцессора
`-E`	Просто вывести (на экран) обработанный препроцессором исходный код

`-I` *директория* — Добавить директорию в путь поиска включаемых файлов и модулей. Поиск осуществляется в следующем порядке: директория, где расположен главный исходный файл, текущая директория, директория, заданная через `-I`, директории, заданные в переменной окружения `G95_INCLUDE_PATH` и в конце концов – в системных директориях.

Инструкции, имеющие отношение к Фортрану

`-Wall` — Включить большинство предупреждений.

`-Werror` — Заменять предупреждения на ошибки.

`-Werror=`*номера* — Заменять список заданных через запятую предупреждений на ошибки.

`-Wextra` — Включить предупреждения, не включенные инструкцией `-Wall`. `-Wobsolescent`, `-Wunused-module-vars`, `-Wunused-module-procs`, `-Wunused-internal-procs`, `-Wunused-parameter`, `-Wunused-types`, `-Wmissing-intent` и `-Wimplicit-interface`.

`-Wglobals` — Перекрестная проверка использования и декларации процедур внутри одного и того же исходного файла. Включена по умолчанию, но может быть отменена с помощью `-Wno-globals`.

`-Wimplicit-none` — То же, что и `-fimplicit-none`.

`-Wimplicit-interface` — Предупреждать при использовании неявного интерфейса.

`-Wline-truncation` — Предупреждать об обрезании строк кода.

`-Wmissing-intent` — Предупреждать о неуказанных атрибутах назначения формальных аргументов.

`-Wobsolescent` — Предупреждать об использовании устаревающих конструкций.

`-Wno=`*номера* — Отключить предупреждения, заданные через запятую.

`-Wuninitialized` — Предупреждать об использовании неинициализированных (с незаданным значением) переменных. Требует `-O2`.

`-Wunused-internal-procs` — Предупреждать о никогда неиспользованных внутренних процедурах.

`-Wunused-vars` — Предупреждать о неиспользованных переменных.

`-Wunused-types` — Предупреждать о неиспользованных модульных типах. Не включено в `-Wall`.

`-Wunset-vars` — Предупреждать о незаданных переменных.

`-Wunused-module-vars` — Предупреждать о неиспользованных модульных переменных. Полезно для перечисления переменных в `ONLY` списках.

`-Wunused-module-procs` — Предупреждать о неиспользованных модульных процедурах. Полезно для перечисления переменных в `ONLY` списках.

`-Wunused-parameter` — Предупреждать о неиспользованных параметрах. Не включено в `-Wall`.

`-Wprecision-loss` — Предупреждать о потере точности при неявных преобразованиях между разными типами данных.

`-fbackslash` — Интерпретировать обратный слэш, как специальный символ для управляющих последовательностей символов (escape codes). Эта опция включена по умолчанию. Используйте `-fno-backslash`, чтобы интерпретировать обратные слеши буквально.

`-fc-binding` — Вывести Си-прототипы процедур на стандартный вывод.

`-fd-comment` — Превратить D-строки в исполнимые операторы в фиксированном формате.

`-fdollar-ok` — Допускать символ $ в именах.

`-fendian=`*значение* — Задать порядок распределения битовых полей (endian-ness) для неформатных записи и чтения. *значение* должно быть либо `big`, либо `little`. Аннулирует значения, заданные в переменных окружения, используемых при исполнении.

`-ffixed-form` — Считать исходный код написанным в фиксированном формате.

`-ffixed-line-length-132` — Задать длину строки в 132 символа для фиксированного формата.

`-ffixed-line-length-80` — Задать длину строки в 80 символов для фиксированного формата.

`-ffree-form` — Считать исходный код написанным в свободном формате.

`-ffree-line-length-huge` — Принимать очень длинные строки (10 тыс. символов).

`-fimplicit-none` — Запретить декларацию типов переменных по умолчанию, за исключением случаев явного использования оператора `IMPLICIT`.

`-fintrinsic-extensions` — Допускать использование специальных встроенных в g95 функций даже при использовании инструкции `-std=`.

`-fintrinsic-extensions=` Включить выбранные встроенные в g95 функции даже при использовании инструкции `-std=`. Список разделяется запятыми, имена нечувствительны к регистру.

`-fmod=`*директория* Поместить файлы модулей в *директорию*.

`-fmodule-private` Задать доступ по умолчанию к объектам модулей как `PRIVATE`.

`-fmultiple-save` Позволять атрибуту `SAVE` использоваться несколько раз.

`-fone-error` Безусловно прекратить компиляцию после первой ошибки.

`-ftr15581` Разрешить использование расширений TR15581 для динамически размещаемых массивов даже в `-std=F` и `-std=f95` режимах.

`-std=F` Предупреждать об использовании нестандартных для F конструкций. См. `http://www.fortran.com/F`.

`-std=f2003` Строгая проверка на соответствие стандарту Fortran 2003.

`-std=f95` Строгая проверка на соответствие стандарту Fortran 95.

`-i4` Задать целочисленный тип без спецификации (т.е. с неуказанным параметром kind) как kind=4 (32 бита).

`-i8` Задать целочисленный тип без спецификации как kind=8 (64 бита).

`-r8` Задать вещественный тип без спецификации как kind=8 (64 бита, “двойная точность”).

`-d8` Задает `-i8` и `-r8`.

Инструкции для генерации бинарного кода

`-fbounds-check` Проверять границы массивов и строк во время исполнения.

`-fcase-upper` Перевести все общедоступные символы (public symbols) в верхний регистр (заглавные буквы).

`-fleading-underscore` Добавить знак нижнего подчеркивания к общедоступным именам.

`-fonetrip` Исполнять циклы `DO` по крайней мере один раз. (Из древнего `FORTRAN` 66).

`-fpack-derived` Постараться разместить данные производных типов компактнее. Требует меньше памяти, но может замедлить программу.

`-fqkind=`*n* Задать тип вещественный данных с ’q’ экспонентой как *n*.

`-fsecond-underscore` Добавить второй знак нижнего подчеркивания к именам с одним таким знаком (эта используется по умолчанию). Используйте `-fno-second-underscore` для отмены.

`-fshort-circuit` Не вычислять значение второй половины операторов `.AND.` и `.OR.` если значение уже известно из первой половины.

`-fsloppy-char` Не выводить ошибки записи несимвольных данных в символьных операциях и разрешить сравнения между целочисленными (`INTEGER`) и символьными (`CHARACTER`) данными.

`-fstatic` Разместить локальные переменные в статической памяти по мере возможности. Это НЕ то же самое, что и линковка всего статически (`-static`).

`-ftrace=` `-ftrace=frame` Добавляет код для отслеживания ошибок при незапланированной остановке программы. Эта опция замедляет программу. `-ftrace=full` добавляет возможность проследить ошибку обратно в код с указанием номера строки (еще медленее). По умолчанию задано `-ftrace=none`.

`-funderscoring` Добавить знак нижнего подчеркивания в конце глобальных имен. Эта опция включена по умолчанию, используйте `-fno-underscoring` для отмены.

`-max-frame-size=`*n* Задать максимальный размер (в байтах) стекового фрейма до того, как массивы размещаются динамически.

`-finteger=`*n* Инициализировать неинициализированные целочисленные скалярные переменные значением *n*.

`-flogical=`*значение* Инициализировать неинициализированные скалярные переменные логического типа. Допустимые *значения* – `none`, `true` и `false`.

`-freal=`*число* Инициализировать неинициализированные скалярные переменные вещественного и комплексного типов. Допустимые *числа* – `none`, `zero`, `nan`, `inf`, `+inf` и `-inf`.

`-fpointer=`*значение* Инициализировать скалярные ссылки. Допустимые *значения* – `none`, `null` и `invalid`.

`-fround=`*значение* Настройка способа округления во время компиляции. *значение* может быть `nearest`, `plus`, `minus` и `zero`. По умолчанию округляется до ближайшего значения, `plus` округляет до плюс бесконечности, `minus` до минус бесконечности, `zero` до нуля.

`-fzero` Инициализировать численные переменные как ноль, логические как НЕТ (false) и ссылки как "не ассоциирована с адресатом" (null). Другие способы инициализации преобладают над данным.

Инструкции для используемых директорий

`-I` *директория* Добавить *директорию* в путь поиска включаемых файлов и модулей.

`-L` *директория* Добавить *директорию* в путь поиска библиотек.

`-fmod=`*директория* Поместить модули в *директорию*

Переменные окружения

Существует много способов для тонкой настройки поведения вашей программы во время ее исполнения. Это реализуется через "переменные окружения" (environment variables). Запустите программу, скомпилированную g95 с инструкцией `-g95`, чтобы вывести описание всех этих переменных на стандартный вывод. Эти переменные всегда имеют символьные значения, но значения интерпретируются, как целочисленные или логические. Только первый символ логических величин имеет значение и должен быть '`t`', '`f`', '`y`', '`n`', '`1`' или '`0`' (заглавные буквы тоже подойдут). Если значение задано неправильно, то сообщение об ошибке не выдается и используется значение по умолчанию. Для информации о переменных окружения GCC, используемых g95, таких как `LIBRARY_PATH`, обращайтесь к документации по GCC.

`G95_STDIN_UNIT`	целое	Номер устройства ввода/вывода, используемого в качестве стандартного ввода. Отрицательные значения не принимаются, по умолчанию используется 5.
`G95_STDOUT_UNIT`	целое	Номер устройства ввода/вывода, используемого в качестве стандартного вывода. Отрицательные значения не принимаются, по умолчанию используется 6.
`G95_STDERR_UNIT`	целое	Номер устройства ввода/вывода, используемого в качестве стандартного канала ошибок. Отрицательные значения не принимаются, по умолчанию используется 0.
`G95_USE_STDERR`	логическое	Посылает сообщения библиотек на стандартный канал ошибок вместо стандартного вывода. Да – по умолчанию.
`G95_ENDIAN`	символьное	Порядок распределения битовых полей (endian-ness) для неформатного ввода/вывода. Бывает `BIG`, `LITTLE` или `NATIVE`. По умолчанию задано `NATIVE`.
`G95_CR`	логическое	Возврат каретки во время вывода последовательных форматных записей. По умолчанию TRUE на не-Cygwin/Windows, FALSE во всех остальных случаях.
`G95_INPUT_CR`	логическое	Интерпретировать возврат каретки, как часть записи, а не как перевод строки. По умолчанию – TRUE.
`G95_IGNORE_ENDFILE`	логическое	Игнорировать попытки чтения за пределами записи ENDFILE в режиме последовательного доступа. По умолчанию – FALSE.
`G95_TMPDIR`	символьное	Директория для временных файлов. Используется вместо переменной окружения `TMP`. Если `TMP` не определено, то используется `/var/tmp`. Нет значения по умолчанию.
`G95_UNBUFFERED_ALL`	логическое	Если TRUE, то весь вывод небуферизован. Это замедлит запись больших объемов данных, но полезно для мгновенного (конечно же, относительно) появления выходных данных. По умолчанию – FALSE.
`G95_SHOW_LOCUS`	логическое	Если TRUE, то выводит имя файла и номер строки, где произошла ошибка. По умолчанию – TRUE.
`G95_STOP_CODE`	логическое	Если TRUE, то код выхода командой `stop` передается системе как обычный код выхода (exit code). По умолчанию – TRUE.
`G95_OPTIONAL_PLUS`	логическое	Выводит (добавляет) необязательный плюс к числам, где это возможно. По умолчанию – FALSE.

G95_DEFAULT_RECL	целое	Длина записи по умолчанию для файлов последовательного доступа. Наиболее полезно для подстройки значений длины строки присоединенных устройств ввода/вывода. По умолчанию – 50000000.
G95_LIST_SEPARATOR	символьное	Разделитель для вывода под управлением списка переменных. Может содержать любое количество пробелов и максимум одну запятую. По умолчанию – один пробел.
G95_LIST_EXP	целое	Максимальная степень десяти, которая не использует формат с экспонентой (например 1E+02). По умолчанию – 6.
G95_COMMA	логического	Использовать запятую как разделитель в десятичных дробях во время ввода/вывода. По умолчанию – FALSE.
G95_EXPAND_UNPRINTABLE	логическое	Для форматного вывода выводить непечатные символы в виде обозначений с обратным слэшем (\). По умолчанию FALSE.
G95_QUIET	логическое	Не использовать звуковой системный сигнал (\a) в форматном выводе. По умолчанию – FALSE.
G95_SYSTEM_CLOCK	целое	Число замеров в секунду, выдаваемое командой SYSTEM_CLOCK(). Ноль (0) отключает часы. По умолчанию – 100000.
G95_SEED_RNG	логическое	Если TRUE, задавать новое начальное число для генератора случайных чисел при каждом запуске программы. По умолчанию – FALSE.
G95_MINUS_ZERO	логическое	Если TRUE, выводит нулевые значения без знака минус при форматном выводе, даже если внутреннее значение является отрицательным или -0. Это традиционный, но нестандартный способ вывода нулей. По умолчанию – FALSE.
G95_ABORT	логическое	Если TRUE, создает corefile при аномальной остановке программы. Полезно для нахождения местоположения проблемы. По умолчанию – FALSE.
G95_MEM_INIT	символьное	Как инициализировать динамически размещенную память. По умолчанию используется NONE для отсутствия инициализации (работает быстрее), NAN для инициализации как Не-Число (Not-a-Number) с мантиссой 0x00f95 или любым другим шестнадцатиричным числом.
G95_MEM_SEGMENTS	целое	Максимальное число неосвобожденных областей памяти, о которых сообщает программа при выходе. 0 означает 0, меньше 0 – показать все. По умолчанию – 25.
G95_MEM_MAXALLOC	логическое	Если TRUE, показывает максимальное число байтов памяти, использованной в пользовательком адресном пространстве, в течение исполнения. По умолчанию – FALSE.
G95_MEM_MXFAST	целое	Максимальный размер запроса для передачи запроса из “быстрых” ячеек. “Быстрые” ячейки работают быстрее, но больше подвержены фрагментации. По умолчанию – 64 байта.
G95_MEM_TRIM_THRESHOLD	целое	Размер “самой верхней” (top-most) памяти, которая остается занятой, пока не возвращается операционной системе. -1 предотвращает возвращение памяти системе. Полезно для программ с долгим временем исполнения. По умолчанию – 262144.
G95_MEM_TOP_PAD	целое	Дополнительный размер для размещения при получении памяти от ОС. Может ускорить последующие запросы. По умолчанию – 0.
G95_SIGHUP	символьное	Что делать программе при получении сигнала SIGHUP – IGNORE, ABORT, DUMP или DUMP-QUIT. По умолчанию – ABORT. Только на Unix.
G95_SIGINT	символьное	Что делать программе при получении сигнала SIGINT – IGNORE, ABORT, DUMP или DUMP-QUIT. По умолчанию – ABORT. Только на Unix.
G95_SIGQUIT	символьное	Что делать программе при получении сигнала SIGQUIT – IGNORE, ABORT, DUMP или DUMP-QUIT. По умолчанию – ABORT. Только на Unix.

G95_CHECKPOINT	целое	На x86 Linux, количество секунд между контрольными точками создания corefile. Ноль означает не создавать corefile.
G95_CHECKPOINT_MSG	логическое	Если TRUE, выводить сообщение на стандартный канал ошибок, когда создана контрольная точка corefile. По умолчанию – TRUE.
G95_FPU_ROUND	символьное	Задать режим округления операций с плавающей точкой. Может быть NEAREST, UP, DOWN, ZERO. По умолчанию – NEAREST.
G95_FPU_PRECISION	символьное	Точность промежуточных результатов. Может иметь значения 24, 53 and 64. По умолчанию – 64. Работает только на x86 и совместимых.
G95_FPU_DENORMAL	логическое	Выдать исключение (exception), когда получено аномальное число. По умолчанию – FALSE.
G95_FPU_INVALID	логическое	Выдать исключение (exception), при выполнении неправильной операции. По умолчанию – FALSE.
G95_FPU_ZERODIV	логическое	Выдать исключение (exception), при делении на ноль. По умолчанию – FALSE.
G95_FPU_OVERFLOW	логическое	Выдать исключение (exception), при переполнении (overflow). По умолчанию – FALSE.
G95_FPU_UNDERFLOW	логическое	Выдать исключение (exception), при потере значимости (underflow). По умолчанию – FALSE.
G95_FPU_INEXACT	логическое	Выдать исключение (exception), при потере точности. По умолчанию – FALSE.
G95_FPU_EXCEPTIONS	логическое	Нужно ли показывать скрытые исключения (exceptions) на выходе из программы. По умолчанию – FALSE.
G95_UNIT_x	символьное	Изменяет имя по умолчанию для устройства ввода/вывода x. По умолчанию – fort.x
G95_UNBUFFERED_x	логическое	Если TRUE, устройство ввода/вывода x не буферизуется. По умолчанию – FALSE.

Коды ошибок исполнения

Запуск программы, скомпилированной с g95, с опцией -g95 выведет список кодов ошибок на стандартный вывод.

-2 Конец записи
-1 Конец файла
0 Успешное завершение
Коды ошибок операционной системы (1 - 199)
200 Конфликтующие параметры оператора
201 Неправильные параметры оператора
202 Пропущенный параметр оператора
203 Файл уже был открыт под другим номером устройства ввода/вывода
204 Неоткрытое устройство ввода/вывода
205 Ошибка в формате (FORMAT)
206 Неправильно указанная операция над файлом (ACTION)
207 Чтение за пределами конца файла (ENDFILE record)
208 Получено неподходящее значение во время чтения
209 Численное переполнение во время чтения
210 Недостаточно памяти
211 Массив уже размещен в памяти
212 Освобождена некорректная ссылка
214 Поврежденная запись при неформатном доступе к файлу
215 Попытка считать больше данных, чем длина записи (RECL)
216 Попытка записать больше данных, чем длина записи (RECL)

Функциональные возможности стандарта Fortran 2003

В G95 реализована часть функционала стандарта Fortran 2003. Получить информацию о свойствах Fortran 2003 можно тут: http://www.kcl.ac.uk/kis/support/cit/fortran/john_reid_new_2003.pdf.

- Реализованы следующие встроенные процедуры: COMMAND_ARGUMENT_COUNT(), GET_COMMAND_ARGUMENT(), GET_COMMAND() и GET_ENVIRONMENT_VARIABLE()

- Переменные цикла DO вещественного типа (включая двойную точность) не поддерживаются в g95.
- Квадратные скобки [и] могут использоваться как альтернатива (/ и /) в конструкторах массивов.
- TR 15581 – динамически размещаемые компоненты производных типов. Использование динамически размещаемых `ALLOCATABLE` атрибутов формальных параметров, результатов вычисления функций и компонентов структур.
- Потоковый (Stream) ввод/вывод – F2003 потоковый доступ дает возможность читать и писать бинарные файлы, не заботясь о структуре записей. Клайв Пейдж (Clive Page) написал документацию на эту тему, см.:
  `http://www.star.le.ac.uk/~cgp/streamIO.html`.
- Оператор `IMPORT`. Используется в теле интерфейса для доступа к объектам объемлющей контекстной области (Host Scoping Unit).
- "Европейский" способ представления вещественных чисел – тэг `DECIMAL='COMMA'` используется в операторах `OPEN`, `READ` и `WRITE` для отделения десятичной части дробей запятой вместо точки.
- `MIN()` и `MAX()` работают с символьными данными.
- Атрибут задания типа `VALUE` для формальных параметров подпрограммы передает фактический аргумент по значению.
- Поддерживаются конструкции структур в стиле F2003.
- Поддерживаются указатели на процедуры.
- Оператор `BIND(C)`, `ISO_C_BINDING` модуль обеспечивают возможность облегченного взаимодействия с Си.

Взаимодействие G95 с другими компиляторами/языками

В то время как g95 создает независимые исполняемые файлы, иногда нужно взаимодействие с другими программами, обычно – на Си. Первая трудность в программировании на нескольких языках – это имена общедоступных символов (public symbols). G95 следует правилам f2c, то есть добавляет знак нижнего подчеркивания к общедоступным именам, или два знака, если имя оканчивается таким знаком. Инструкции `-fno-second-underscore` и `-fno-underscoring` могут быть полезными для контроля за совместимостью с вашим Си компилятором. Используйте программу `nm` для просмотра `.o` файлов, создаваемых обоими компиляторами. Также G95 преобразует общедоступные имена в нижний регистр, если только не дана инструкция `-fupper-case`, в случае чего все будет в верхнем регистре. Имена модулей представляются как *имя-модуля*`_MP_`*имя-языкового-объекта* (*module-name*`_MP_`*entity-name*).

После процесса линковки, может быть 2 случая: вызов Си-процедур из Фортрана и вызов фортрановских процедур из Си. При вызове фортрановских подпрограмм из Си они зачастую вызывают подпрограммы из библиотеки Фортрана, при этом ожидается, что динамическая память (heap) будет инициализирована каким-то образом. Чтобы задать ручную инициализацию из Си, вызовите `g95_runtime_start()` для инициализации фортрановской библиотеки и `g95_runtime_stop()` по окончании. Прототипом `g95_runtime_start()` является:

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

Библиотека должна уметь работать с параметрами командной строки. Если это необязательно, то дайте следующие инструкции `argc=0` и `argv=NULL`. На OSX, задайте `-lSystemStubs` при использовании g95 в качестве сборщика (линкера) и линковке с объектными файлами, созданными GCC.

Фортран 2003 предоставляет возможности для облегченного взаимодействия с Си. Оператор `BIND(C)` создает символы, которые легче использовать из Си (или других языков). Например

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

Эта команда создает символ с именем `foo` без добавления каких-либо лишних знаков (подчеркивания). Все имена создаются в нижнем регистре. Аналогично:

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

Это превращает имя символа в `Foo1`. Из фортрана, эта подпрограмма вызывается, как обычно, через имя `foo`, `FOO` или любую другую комбинацию.

Из программы на Си аргументы передаются по значению, в то время как фортран передает их по ссылке. Фортран 2003 дает возможность использовать атрибут `VALUE` для передачи формальных параметров по значению. Например:

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

Подпрограмма, написанная таким образом, тем не менее, может быть вызвана из фортрана вместе с ограничением, что формальные параметры не ассоциированы с фактическими, и изменение формального параметра не ведет к изменению фактического.

Глобальные переменные могут использованы схожим способом. Следующая подпрограмма выводит на экран величины переменной VAR, которая не была бы доступна в фортране без данной функции:

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

В то время, как Фортран различает типы данных с помощью атрибута kind, Си использует различные ключевые слова. Для того, чтобы использовать один и тот же объект, Фортран 2003 предоставляет встроенный модуль ISO_C_BINDING, который содержит информацию о преобразовании данных из Фортрана в Си. Когда этот модуль используется с помощью USE, задаются следующие параметры (PARAMETER):

c_int	Целочисленный тип Си int
c_short	Целочисленный тип Си short
c_long	Целочисленный тип Си long
c_long_long	Целочисленный тип Си long long
c_signed_char	Целочисленный тип Си char
c_size_t	Целочисленный тип Си size_t
c_intptr_t	Целочисленный тип Си, такой же, как ссылки в Си
c_float	Вещественный тип Си float
c_double	Вещественный тип Си double

Модуль ISO_C_BINDING содержит много других функций. Используя его, можно написать программу типа:

```
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
    ...
```

Использование генератора случайных чисел

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

Возвращает вещественный (REAL) скаляр или массив вещественных (REAL) случайных чисел как результат, $0 \leq$ harvest < 1.

Задание начальной точки генератора:

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

sz - это минимальное число целых чисел, нужных для задания начальной точки; g95 возвращает 4. Аргумент pt является массивом, состоящим из целых чисел (целочисленного типа по умолчанию), размера $n1 \geq$ sz, и содержит заданные пользователем значения "затравки"(seed). Аргумент gt является массивом целых чисел размером $n2 \geq$ sz, и содержит текущую затравку.

Вызов RANDOM_SEED() без аргументов осуществляет затравку со значением, определяемым текущим моментом времени. Это можно использовать для генерации случайных последовательностей, которые будут отличаться при каждом запуске программы. Затравка инициализируется значением, вычисляемым на основе системного времени, если переменная G95_SEED_RNG равна TRUE. Если ни одно из этих условий не выполняется, то RANDOM_NUMBER() всегда выдает одну и ту же последовательность.

Используется xor-shift генератор, разработанный Джорджем Марсаглией (George Marsaglia).

Заданные макросы препроцессора

Следующие макросы определены всегда:

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

Условные макросы:

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

Функция возобновления исполнения из corefile

На системах x86 Linux исполнение программы, скомпилированной g95, может быть приостановлено, а затем продолжено с места остановки. Если вы прерываете исполнение с помощью сигнала QUIT, обычно с помощью control-backslash, программа создаст исполняемый файл с именем `dump` в текущей директории. Запустив этот файл, вы продолжите исполнение вашей программы с того момента, когда она была остановлена. Пример:

```
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      (тут нажимается control-\)
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 %
```

Все открытые файлы должны находится в тех же самых местах, где и раньше. Если вы линкуете с другими языками, то эта функция, возможно, не будет работать. В то время, как главная цель этого – сохранить данные во время перезапуска системы, можно также использовать для того, чтобы прогнать длинную задачу в несколько коротких этапов или перенести исполнение на другой компьютер. Автоматическое создание контрольных точек может быть задано с помощью переменной `G95_CHECKPOINT` в секундах между двумя точками. Ноль значит “не использовать эту функцию”. Каждый последующий файл записывается на место предыдущего.

“Умная” компиляция

Рассмотрим модуль `foo` из файла `foo.f95`. Есть два типа изменений, которые могут быть произведены с файлом `foo.f95`:

1. Такие, что меняют то, как используется модуль `foo`, например, изменяется интерфейс процедуры;
2. Такие, что НЕ меняют то, как используется модуль `foo`, но меняют реализацию, например, исправлена ошибка в теле процедуры.

Оба типа обычно меняют содержание объектного файла `foo.o`, но только первый тип изменяет содержание файла модуля foo.mod. Во время компиляции g95 достаточно “умен”, чтобы распознать, требует ли изменения файл `.mod`, то есть при изменениях второго типа старый файл `.mod` остается неизменным.

Эта функция предотвращает ненужную каскадную перекомпиляцию (файлов, зависящих от `foo.mod`) при сборке большой программы. Конечно, допустим, что многие исходные файлы зависят от `foo.mod` прямо (через оператор `USE FOO`) или косвенно (используя модуль, который использует модуль `foo`). Изменение первого типа файла `foo.f95` запустит перекомпиляцию всех зависимостей; к счастью, такие изменения обычно случаются не часто. Более частые изменения второго типа вызовут перекомпиляцию только `foo.f95`, после чего новый объектный файл `foo.o` может быть слинкован с другими (уже существующими) объектными файлами в итоговый исполняемый файл.

Встроенные в G95 расширения (функции)

ACCESS

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

Проверяет, есть ли доступ к файлу в указанном режиме, где “режим” может состоять из одной или более букв из следующего списка `rwxRWX`. Возвращает 0, если права доступа в порядке, и не-0 в противном случае.

ALGAMA

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

Возвращает натуральный логарифм $\Gamma(x)$. `ALGAMA` – родовая функция любого вещественного типа.

BESJ0

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

Возвращает функцию Бесселя первого рода нулевого порядка. Эта функция является родовой.

BESJ1

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

Возвращает функцию Бесселя первого рода первого порядка. Эта функция является родовой.

BESJN

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

Возвращает функцию Бесселя первого рода n-го порядка. Эта функция является родовой.

BESY0

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

Возвращает функцию Бесселя второго рода нулевого порядка. Эта функция является родовой.

BESY1

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

Возвращает функцию Бесселя второго рода первого порядка. Эта функция является родовой.

BESYN

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

Возвращает функцию Бесселя второго рода n-го порядка. Эта функция является родовой.

CHMOD

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

Изменяет unix-права доступа к файлу. Возвращает не-0, если произошла ошибка.

DBESJ0

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

Возвращает функцию Бесселя первого рода нулевого порядка.

DBESJ1

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

Возвращает функцию Бесселя первого рода первого порядка.

DBESJN

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

Возвращает функцию Бесселя первого рода n-го порядка.

DBESY0

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

Возвращает функцию Бесселя второго рода нулевого порядка.

DBESY1

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

Возвращает функцию Бесселя второго рода первого порядка.

DBESYN

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

Возвращает функцию Бесселя второго рода n-го порядка.

DCMPLX

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

Двойной точности `CMPLX`, `x` и `y` могут быть любого численного типа.

DERF

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

Возвращает функцию ошибок `x` двойной точности.

DERFC

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

Возвращает дополнительную функцию ошибок `x` двойной точности.

DFLOAT

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

Перевести `x` в двойную точность. То же, что и `DBLE`.

DGAMMA

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

Возвращает аппроксимацию $\Gamma(x)$.

DLGAMA

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

Возвращает натуральный логарифм $\Gamma(x)$.

DREAL

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

Перевести `x` в двойную точность. То же, что и `DBLE`.

DTIME

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

Передает в элемент `tarray(1)` число истекших секунд пользовательского времени в текущем процессе с момента, когда функция `DTIME` была вызвана в последний раз. Передает в элемент `tarray(2)` число истекших секунд системного времени в текущем процессе с момента, когда функция `DTIME` была вызвана в последний раз. Возвращает сумму двух элементов.

ERF

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

Возвращает функцию ошибок `x`. Эта функция является родовой.

ERFC

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

Возвращает дополнительную функцию ошибок `x`. Эта функция является родовой.

ETIME

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

Передает в элемент `tarray(1)` число истекших секунд пользовательского времени в текущем процессе с момента запуска программы. Передает в элемент `tarray(2)` число истекших секунд системного времени в текущем процессе с момента запуска программы. Возвращает сумму двух элементов.

FNUM

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

Возвращает дескриптор файла, соответствующего устройству ввода/вывода `unit`. Возвращает -1 если устройство не присоединено.

FSTAT

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

Запрашивает данные об открытом файле (в устройстве `unit`) и заполняет массив `sarray()`. Значения в массиве выделяются из структуры, возвращаемой fstat(2), т.е.: `sarray(1)` Номер устройства B/B, `sarray(2)` Номер узла, `sarray(3)` режим открытия файла, `sarray(4)` число ссылок, `sarray(5)` uid владельца, `sarray(6)` gid владельца, `sarray(7)` Тип устройства, `sarray(8)` Размер файла, `sarray(9)` Время доступа, `sarray(10)` Время модификации, `sarray(11)` Время изменения, `sarray(12)` Размер блока, `sarray(13)` Размещенные блоки.

FDATE

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

Возвращает текущую дату и время в формате: Day Mon dd hh:mm:ss yyyy.

FTELL

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

Возвращает текущую позицию в файле `unit` или -1 если `unit` не открыт.

GAMMA

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

Возвращает аппроксимацию $\Gamma(x)$. `GAMMA` является родовой для любого вещественного типа.

GETCWD

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

Возвращает имя текущей рабочей директории в строке `name`. Или не-0 в случае ошибки.

GETGID

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

Возвращает group id текущего процесса.

GETPID

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

Возвращает process id текущего процесса.

GETUID

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

Возвращает user's id.

HOSTNM

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

Присваивает переменной `name` имя хоста (system's host name). Возвращает не-0 в случае ошибки.

IARGC

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

Возвращает число аргументов командной строки, включая саму программу.

ISATTY

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

Возвращает `.true.` тогда и только тогда, когда устройство B/B `unit` является терминалом.

ISNAN

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

Возвращает`.true.`, если `x` не является числом (NaN).

LINK

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

Создает жесткую ссылку между `path1` и `path2`.

LNBLNK

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

То же, что и стандартная функция `len_trim`. Возвращает порядковый номер последнего символа в строке, не являющегося пробелом.

LSTAT

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

Если `file` является символьной ссылкой, то возвращает данные о ссылке. См. `FSTAT()` для дальнейшей информацией. Возвращает не-0 в случае ошибки.

RAND

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

Возвращает квазислучайное число, которое удовлетворяет условию $0 \leq$ `rand` < 1. Если `x` равен 0, то возвращает следующее число в последовательности. Если `x` равен 1, генератор перезапускается с помощью вызова `srand(0)`. Если `x` равен чему-либо другому, то оно используется в качестве нового начального числа с srand.

SECNDS

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

Возвращает местное время с полуночи в секундах минус значение `t`. Эта функция является родовой.

SIGNAL

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

Интерфейс unix-функции `signal`. Возвращает не-0 в случае ошибки.

SIZEOF

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

Аргумент `object` является именем выражения или типа. Возвращает размер в байтах.

STAT

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

Запрашивает данные о файле `file` и помещает их в массив `sarray`. См. функцию `fstat()` для дополнительной информации. Возвращает не-0 в случае ошибки.

SYSTEM

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

Осуществляет вызов внешней команды `cmd`. Возвращает системный код выхода.

TIME

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

Возвращает текущее время в стиле UNIX-функции `time`.

UNLINK

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

Удалить файл/ссылку `file`. Возвращает не-0 в случае ошибки.

%VAL()
Если применяется к переменной в списке формальных аргументов, то передает переменную по значению. Эта псевдофункция не рекомендуется к использованию и реализована только в целях совместимости. Оператор `VALUE` (стандарт Фортран 2003) является стандартным способом для этой цели.

%REF()
Когда применяется к переменной в списке формальных аргументов, то передает переменную по ссылке.

Встроенные в G95 расширения (подпрограммы)

ABORT

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

Выходит из программы и пишет core dump путем отсылания сигнала SIGABORT самому себе (unix).

CHDIR

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

Задает текущую рабочую директорию `dir`.

DTIME

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

Передает в элемент `tarray(1)` число истекших секунд пользовательского времени в текущем процессе с момента, когда функция `DTIME` была вызвана в последний раз. Передает в элемент `tarray(2)` число истекших секунд системного времени в текущем процессе с момента, когда функция `DTIME` была вызвана в последний раз. result – сумма двух элементов.

ETIME

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

Передает в элемент `tarray(1)` число истекших секунд пользовательского времени в текущем процессе с момента запуска программы. Передает в элемент `tarray(2)` число истекших секунд системного времени в текущем процессе с момента запуска программы. result – сумма двух элементов.

EXIT

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

Выходит из программы со статусом `code` после закрытия всех устройств В/В. Это общая подпрограмма.

FDATE

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

Присваивает `date` значение текущей даты в формате: Day Mon dd hh:mm:ss yyyy.

FLUSH

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

Выводит все, что доступно на данный момент, на устройство `unit`, открытое для вывода.

FSTAT

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

Запрашивает данные об открытом файле на устройстве `unit` и заполняет массив `sarray()`. Присваивает не 0 переменной status в случае ошибки. См. справку по `fstat` для информации по массиву `sarray`.

GETARG

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

Присваивает значение `value` `pos`-тому аргументу командной строки.

GETENV

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

Получает значение переменной окружения `variable` и присваивает ее значение переменной `value`.

GETLOG

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

Возвращает login name процесса `name`.

IDATE

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

Присваивает `m` значение текущего месяца, `d` текущего дня месяца и `y` текущего года. Эта подпрограмма не является системно независимой. Используйте стандартную `DATE_AND_TIME`.

LSTAT

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

Если `file` является символьной ссылкой, то возвращает данные о ссылке. См. `FSTAT()` для дальнейшей информацией. Возвращает не-0 в случае ошибки.

RENAME

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

Переименовывает файл `path1` в `path2`. Если используется аргумент `status`, то он принимает ненулевое значение при ошибке.

SIGNAL

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

Интерфейс unix-функции `signal`. Возвращает не-0 в случае ошибки.

SLEEP

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

Делает паузу на `seconds` секунд.

SRAND

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

Заново инициализирует генератор случайных чисел. См. `srand()`.

STAT

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

Запрашивает данные о данном файле `file` и помещает их в массив `sarray`. См. функцию `fstat()` для дополнительной информации. Возвращает не-0 в случае ошибки.

SYSTEM

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

Осуществляет вызов внешней команды `cmd`. Возвращает системный код выхода через `result`, если он задан.

UNLINK

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

Убрать ссылку (удалить) `file`. Возвращает не-0 в случае ошибки.