G95

It's Free Crunch Time

http://www.g95.org

Klíčové rysy G95

- Volně šiřitelný překladač jazyka Fortran 95.
- Současná (September 2006) verze g95 je 0.91.
- GNU Open Source, GPL licence.
- Běh přeložených programů lze modifikovat pomocí mnoha proměnných prostředí, dokumentovaných v samotném přeloženém programu.
- TR15581– Alokovatelné argumenty procedur, komponenty struktur a návratové hodnoty funkcí.
- F2003 procedurální ukazatele, konstruktory struktur, interoperabilita s jazykem C
- F2003 vestavěné procedury a moduly.
- Atribut VALUE pro předávání argumentů procedur hodnotou.
- Volba COMMA v příkazech OPEN, READ, a WRITE pro desetinnou čárku.
- Hranaté závorky `[ ]` jako alternativa k `(/ /)` pro konstruktory polí.
- Příkaz `IMPORT`, použitelný v INTERFACE pro přístup k vnějším deklaracím.
- `MIN()` a `MAX()` pro znakové i numerické typy.
- `OPEN` pro binární proudové I/O.
- Zpětná kompatibilita s g77 Application Binary Interface (ABI).
- Defaultní integer 32 nebo 64 bitů.
- Procedura `SYSTEM()` pro přístup k příkazové řádce.
- Tabulátory v zdrojovém kódu jsou povoleny.
- Volba pro symbolická jména s $.
- Fromát řežězců Hollerith.
- Datový typ DOUBLE COMPLEX.
- Proměnná délka pojmenovaných `COMMON` bloků.
- Míchání řetězcových a číselných typů v `COMMON` a `EQUIVALENCE`.
- `INTEGER` kind = 1, 2, 4, 8.
- `LOGICAL` kind = 1, 2, 4, 8.
- `REAL` kind = 4, 8.
- `REAL(KIND=10)` pro x86-kompatibilní systémy. 19 desetinných číslic, rozsah do $10^{\pm 4931}$.
- Write s hvězdičkovým formátem používá minimální počet číslic potřebný pro jednoznačné rozlišení čísel.
- VAX ladicí (D) řádky.
- Volba pro řetězcové konstanty v C-stylu (např. 'hello\nworld').
- I/O deskriptory \ a $.
- VAX systémové vestavěné procedury (SECNDS atd.)
- Unixové funkce (getenv, etime, stat, etc.)
- Detekce nekonformních nebo nealokovatelných polí za běhu - viz Table IV na: `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`
- Detekce memory leaks - viz Table V na: `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt`
- Trasování běhových chyb.
- Chytrá kompilace bránící řetězové rekompilaci.
- Volba pro F kompatibilitu. Viz `http://www.fortran.com/F`. G95 lze přeložit jako překladač F.
- suspend/resume schopnost programů přeložených pod x86/Linux.
- Zastaralé reálné indexy DO cyklů NEJSOU podporovány.
- Typická je rychlá odezva na zprávy o chybách.
- Lze přeložit s GCC 4.0.3 a 4.1.1.
- Ke stažení pro Linux/x86, PowerPC, 64-bit Opteron, 64-bit Itanium, 64-bit Alpha.
- Ke stažení pro Windows/Cygwin, MinGW, & Interix.
- Ke stažení pro OSX on Power Mac G4, x86-OSX.
- Ke stažení pro FreeBSD na x86, HP-UX 11, Sparc-Solaris, x86-Solaris, OpenBSD, NetBSD, AIX, IRIX, Tru64 UNIX na Alpha.
- Také "Fink" verze.
- Stabilní a vývojové binárky pro většinu platforem ke stažení na `http://ftp.g95.org`.

Sem tam potkám někoho s kým jsem si mailoval o g95. Nejčastěji hovoří o tom, jak výjimečnou práci dělám zcela sám. Vždycky se směju a poukazuji na to, že jsem ji nikdy nedělal sám. S g95 aktivně pomohlo tak okolo tisíce lidí. Někteří si myslí, že kdo píše kód, dělá všechnu práci, ale ve skutečnosti lidé, kteří izolují chyby v překladači na tucet řádků, odvádějí extrémně důležitou práci, která se často přehlíží. Programování něčeho tak složitého jako je moderní překladač Fortranu není práce, kterou byste mohli dělat sami, věřte mi.

Jako mnoho jiného, g95 se zrodilo z frustrace. Já psal kód pro svou doktorskou práci ve Fortranu 77 s pomocí g77. Fortran je úžasný jazyk pro numerické výpočty - rychlý a neučesaný pro ty, kdo se zajímají více o výsledky než psaní programů. Můj kód obsahoval hromadu sofistikovaných datových struktur - spojové seznamy, octree stromy, řídké matice, tvoření sítě pro konečné prvky, řešení Poissonovy rovnice, multipólové rozvoje, minimalizace metodou sdružených gradientů a spoustu výpočetní geometrie. Protože jsem používal Fotran 77, kód byl komplikovaný a šlo by jej hodně vylepšit dynamickou alokací a strukturami. A moje doktorská práce byla minulostí a já potřeboval novou výzvu.

Kromě pohodlí pokročilých jazykových kosntrukcí mě hodně inspirovala práce Billa Kahana. Po přečtení mnoha z jeho článků jsem získal názor, že ačkoli jsou numerické výpočty složité, lze nalézt cesty jak redukovat chyby tak, že už nikoho nezajímají. Uživatel je zde často vydán na milost autorovi knihoven.

Ačkoliv překladač je ta cool část, knihovny mě vždycky zajímaly více. Práce překladače je dost striktně specifikována, ale v knihovnách se mohou rozvíjet inovace a experimenty. I dokud byla v plenkách, bylo v ní hodně ve srovnání s konkurenčními překladači. Třeba schopnost suspend/resume jsem chtěl dlouho předtím, než jsem ji implementoval v g95.

Při psaní g95 jsem si užil hodně zábavy, a těším se na její pokračování v následujících desetiletích.

Andy Vaught
Mesa, Arizona
Říjen 2006

Licence

G95 samotný je uvolněn pod GNU General Public License (GPL). Právní detaily viz: `http://www.gnu.org/licenses/gpl.html`.

Běhové knihovny jsou většinou pod GPL a obsahují výjimku z GPL která dává uživatelům práva používat knihovny g95 s kódy které pod GPL nespadají, a distribuovat programy bez jakýchkoli omezení vyplývajících z GPL.

Instalace

Unix (Linux/OSX/Solaris/Irix/etc.):
Otevřete konzolu, přejděte do adresáře, kam chcete g95 nainstalovat, a spusťe následující příkazy (musíte mít připojení k internetu).

```
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
```

Rozbalí se následující souborová struktura:

```
./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
```

Soubor `cc1` je symbolický odkaz na `f951` ve stejném adresáři.

Cygwin

Volba `-mno-cygwin` umožňuje Cygwiní verzi g95 sestavovat binárky které nepotřebují pro běh přístup ke knihovně cygwin1.dll, a tak je lze spustit na jiných systémech. Také jsou pak programy nezávislé na GNU GPL. Aby volba -mno-cygwin fungovala, musí být nainstalovány knihovny mingw (`http://www.cygwin.com`).
Stáhněte si binárky z `http://ftp.g95.org/g95-x86-cygwin.tgz` do rootovského adresáře Cygwina (obvykle c:\Cygwin). Pusťte Cygwiní konzolu, a zadejte tyto příkazy:

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

Tím nainstalujete g95 do adresáře `/usr/local/bin`. Pozor: Nepoužívejte Winzip na extrakci tar archivu, symbolické odkazy se pak nenastaví správně.

MinGW

Binárky g95 pro prostředí MS-Windows jsou samorozbalovací instalátory. Dostupné jsou momentálně dvě verze: Uživatelé Windows 98 by měli používat balík g95 sestavený gcc 4.0.3, `http://ftp.g95.org/g95-MinGW.exe`. Uživatelé Windows NT, XP a 2000 mohou použít stejný balík, nebo novější sestavený gcc 4.1.1, dostupný na `http://ftp.g95.org/g95-MinGW-41.exe`.

Volně šiřitelný systém MinGW/Msys poskytuje GNU GCC komponenty potřebné pro g95, což zahrnuje linker `ld.exe`, a `as.exe` (GNU assembler) z balíku binutils, dostupného na `http://www.mingw.org`. Instalační skript umožňuje dva druhy instalace. Není-li nalezen MinGW, nainstaluje g95 i s důležitými MinGW

utilitami a knihovnami do složky vybrané uživatelem. Instalační složku zahrňte do proměnné `PATH`, a nastavte proměnnou prostředí `LIBRARY_PATH` na tuto instalační složku.

Máte-li už MinGW nainstalovaný, doporučuje se nainstalovat g95 do kořenového adresáře MinGW (obvykle `C:\mingw`) aby se předešlo případným konfliktům. Pokud instalátor detekuje MinGW, pokusí se do něj g95 nainstalovat. Zahrňte adresář `MinGW\bin` do `PATH`, a nastavte proměnnou

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

Ve Windows 98 a Windows ME to obvykle znamená editovat soubor `autoexec.bat`, a je potřeba restartovat systém.

Poznámka pro uživatele Windows XP: MinGW momentálně dovoluje pouhých 8 MB pro heap. Pokud potřebuje vaše aplikace více paměti, zkuste překládat s: `-Wl,--heap=0x01000000`. Zvětšujte hexadecimální hodnotu `--heap` dokud váš program nepoběží normálně.

Spouštění G95

G95 se rozhoduje jak zkompilovat daný soubor podle jeho přípony. Povolené přípony pro Fortranské zdrojáky jsou `.f`, `.F`, `.for`, `.FOR`, `.f90`, `.F90`, `.f95`, `.F95`, `.f03` a `.F03`. Přípona určuje, zda je soubor ve fixed nebo free formátu. Soubory končící `.f`, `.F`, `.for`, a `.FOR` se předpokládají ve fixed formátu kompatibilním se starými f77 soubory. Soubory končící `.f90`, `.F90`, `.f95`, `.F95`, `.f03` a `.F03` se předpokládají ve free formátu. Soubory s příponou velkými písmeny jsou automaticky před překladem prohnány C preprocesorem, s příponou malým písmem nikoli.

Pozn.překl. nedoporučuje se volit přípony f90,f95,f03 podle verze standardu Fortranu, kterým se hlavně řídíte (zdroj c.l.f. google group). Přípona f90 (poněkud nešťastně zvolená) má označovat, že jde o zdrojový text ve free formátu, zatímco .f zůstane vyhrazeno pro formát fixed.

Základní volby pro překlad Fortranu s g95 jsou:

`-c` Jen přeložit, nesestavovat.

`-v` Vypiš příkazy (programy s argumenty) spouštěné g95. Obzvláště užitečné pro řešení problému s cestami.

`-o` Specifikace jména výstupního souboru, buď objektového kódu nebo spustitelného. Pod windows se automaticky přidá přípona `.exe`, pokud není uvedena. Není-li specifikováno jinak, automatický výstupní spustitelný soubor je `a.out` v unixu, `a.exe` pod Windows.

Jednoduché příklady:

`g95 -c hello.f90`
Přeloží `hello.f90` do objekt kódu `hello.o`.

`g95 hello.f90`
Přeloží `hello.f90` a sestaví jej do spustitelného souboru `a.out` (v unixech), nebo `a.exe` (na systémech MS Windows).

`g95 -c h1.f90 h2.f90 h3.f90`
Přeloží několik zdrojových souborů. Nejsou-li chyby, vytvoří se objektové soubory `h1.o`, `h2.o` a `h3.o`.

`g95 -o hello h1.f90 h2.f90 h3.f90`
Přeloží několik zdrojových souborů a sestaví je do spustitelného souboru `a.out` (v unixech), nebo `a.exe` (na systémech MS Windows).

Syntax voleb

`g95 [ -c | -S | -E ]`	Přelož | vyrob assemblerový kód | vypiš zdroj (po preprocesingu)
`[-g] [-pg]`	Ladicí překlad
`[-O[`*n*`] ]`	Level optimalizací, $n = 0, 1, 2, 3$
`[-s ]`	Ořež debug info
`[-W`*jméno warningu* `] [-pedantic]`	Různá varování
`[-I`*adresář* `]`	Adresář pro USE a INCLUDE

`[-L`*adresář* `]`	Adresář pro knihovny
`[-D makro[=hodnota]... ]`	Definuj makro (pro cpp)
`[-U makro ]`	Oddefinuj makro
`[-f volba ...]`	Obecné volby překladu (viz mauál GCC)
`[-m strojová-volba ...]`	Strojové volby překladu (viz manuál GCC)
`[-o výstupní soubor ]`	Jméno výstupního souboru
`vstupní soubor`	

Volby G95

Užití: g95 [volby] soubor...

`-pass-exit-codes`	Ukonči fázi s nejvyšším chybovým kódem během ní.
`--help`	Zobraz tento help.
`--target-help`	Zobrazí volby příkazové řádky specifické pro cíl překladu. (Použijte '`-v --help`' pro zobrazení voleb podprocesů).
`-dumpspecs`	Zobrazí vestavěné spec řetězce.
`-dumpversion`	Zobrazí verzi překladače.
`-dumpmachine`	Zobrazí cílový procesor překladače.
`-print-search-dirs`	Zobrazí adresáře v překladačem prohledávaných cestách.
`-print-libgcc-file-name`	Zobrazí doprovodnou knihovnu překladače.
`-print-file-name=`*lib*	Zobrazí plnou cestu ke knihovně *lib*.
`-print-prog-name=`*prog*	Zobrazí plnou cestu ke komponentě *prog*.
`-print-multi-directory`	Zobrazí kořenový adresář pro verze libgcc.
`-print-multi-lib`	Zobrazí vztahy mezi volbami přík. řádku a prohledáváním více adresářů.
`-print-multi-os-directory`	Zobrazí relativní cestu k systémovým knihovnám.
`-Wa,`*options*	Předá volby (oddělené čárkami) assembleru.
`-Wp,`*options*	Předá volby (oddělené čárkami) preprocesoru.
`-Wl,`*options*	Předá volby (oddělené čárkami) linkeru.
`-Xassembler` *arg*	Předá argument *arg* assembleru.
`-Xpreprocessor` *arg*	Předá argument *arg* preprocesoru.
`-Xlinker` *arg*	Předá argument *arg* linkeru.
`-save-temps`	Nemazat dočasné soubory.
`-pipe`	Použij roury místo dočasných souborů, je-li to možné.
`-time`	Měř čas běhu podprocesů. Není na některých platformách (MinGW, OSX).
`-specs=`*file*	Nahradí vestavěné specifikace obsahem souboru *file*.
`-std=`*standard*	Předpokládá zdrojové kódy pro daný standard Fortranu.
`-B` *directory*	Přidá adresář *directory* k prohledávané cestě překladače.
`-b` *machine*	Spusť gcc pro cílový stroj *machine*, je-li to možné.
`-V` *version*	Spusť gcc verze *version*, je-li to možné.
`-v`	Zobraz programy spouštěné překladačem.
`-M`	Vypiš závislosti ve stylu Makefile.
`-###`	Jako `-v` ale volby se vypíší a příkazy nevykonávají.
`-E`	Jen preprocesuj; nepřekládej.
`-S`	Překlad jen do assembleru.
`-c`	Přelož, nesestavuj.
`-o` *file*	Výstup do souboru *file*.
`-x` *language*	Specifikuj jazyk *language* následujících vstupních souborů. Povoleno je: c, c++, assembler, none; 'none' znamená návrat k normálnímu rozhodování podle přípony.

Volby začínající `-g`, `-f`, `-m`, `-O`, `-W`, nebo `--param` jsou automaticky předávány spouštěným subprocesům. Pro předání jiných voleb je třeba použít `-W`*letter*. Pro hlášení chyb (bug reporting), viz: `http://www.g95.org`.

Nespecifikujeme-li jinak, g95 překládá bez optimalizací. Číslo *n* volby `-O`*n* udává úroveň optimalizace, od 0 do 3. Nula znamená žádné optimalizace, vyšší čísla vyšší agresivitu optimalizací. V řežimu optimalizace

smí překladač měnit kód za účelem urychlení. Často se tak drobně ovlivní výsledky výpočtů. `-O` je totéž co `-O1`.

Podstatného zrychlení lze dosáhnout zahrnutím alespoň `-O2 -march=`*arch* kde *arch* je architektura vašeho procesoru, čili `pentium4`, `athlon`, `opteron`, atd. Další volby typické pro Fortran zahrnují `-funroll-loops`,`-fomit-frame-pointer`, `-malign-double` a `-msse2`.
Pro informace o všech volbách GCC viz: `http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc`.

Volby preprocesoru

G95 umí zpracovat soubory s kontrukcemi preprocesoru jazyka C.

`-cpp`	Vynutí zpracování zdrojáků preprocesorem.
`-no-cpp`	Zakáže totéž.
`-D name[=value]`	Definuje makro preprocesoru.
`-U name`	Oddefinuje makro preprocesoru.
`-E`	Ukáže výsledek preprocesingu.
`-I` *directory*	Přidá adresář *directory* do cesty prohledávané pro #include. Soubory se hledají v adresářích v tomto pořadí: Adresář hlavního zdrojového souboru, aktuální adresář, adresáře v `-I`, adresáře v proměnné prostředí `G95_INCLUDE_PATH` a nakonec systémové adresáře.

Volby Fortranu

`-Wall`	Zapne většinu varovných hlášení.
`-Werror`	Bere varování chyby.
`-Werror=`*numbers*	Bere (čísly) vybraná varování jako chyby.
`-Wextra`	Zapne dalí varovná hlášení, nezapnutá při `-Wall`. Jmenovitě `-Wobsolescent`, `-Wunused-module-vars`, `-Wunused-module-procs`, `-Wunused-internal-procs`, `-Wunused-parameter`, `-Wunused-types`, `-Wmissing-intent` a `-Wimplicit-interface`.
`-Wglobals`	Křížová kontrola volání a definic procedur ve stejném zdrojovém textu. Normálně zapnuto, `-Wno-globals` vypne.
`-Wimplicit-none`	Totéž co `-fimplicit-none`.
`-Wimplicit-interface`	Varuje při použití implicitních interface.
`-Wline-truncation`	Varuje při ořezání řádků.
`-Wmissing-intent`	Varuje při chybějících INTENTech argumentů.
`-Wobsolescent`	Varuje před zastaralými konstrukcemi.
`-Wno=`*numbers*	Vypne množinu varování podle čísel (oddělených čárkou).
`-Wuninitialized`	Varuje při použití neinicializovaných proměnných. Funguje jen s -O2 nebo vyšším.
`-Wunused-internal-procs`	Varuje, pokud vnořená procedura není nikdy použita.
`-Wunused-vars`	Varuje před nepoužitými proměnnými.
`-Wunused-types`	Varuje před nepoužitými typy v modulech. Nezapíná se při `-Wall`.
`-Wunset-vars`	Varuje před proměnnými, kterým není přiřazena hodnota.
`-Wunused-module-vars`	Varuje před nepoužitými proměnnými modulů. Užitečné pro sestavování `ONLY` klauzulí.
`-Wunused-module-procs`	Varuje před nepoužitými procedurami modulů. Užitečné pro sestavování `ONLY` klauzulí.
`-Wunused-parameter`	Varuje před nepoužitými parametry. Nezapíná se při `-Wall`.
`-Wprecision-loss`	Varuje před ztrátou přesnosti v implicitních typových konverzích.
`-fbackslash`	Interpretuje zpětná lomítka ve znakových konstantách jako escape kódy. Tato volba je defaultně zapnuta. Použijte `-fno-backslash`, chcete-li brát zpětná lomítka jako zpětná lomítka.
`-fc-binding`	Vypiš C prototypy procedur na standardní výstup.
`-fd-comment`	Ve fixed formátu zahrne do kódu řádky začínající D.

`-fdollar-ok`	Povolit dolary ve jménech entit.
`-fendian=`*value*	Nastaví endian neformátovaných čtení a zápisů. *value* musí být `big` nebo `little`. Má přednost před proměnnými prostředí.
`-ffixed-form`	Předpokládá formát fixed pro zdrojové texty.
`-ffixed-line-length-132`	Řádky o 132 znacích ve fixed formátu.
`-ffixed-line-length-80`	Řádky o 80 znacích ve fixed formátu.
`-ffree-form`	Předpokládá volný formát pro zdrojové texty.
`-ffree-line-length-huge`	Povolit velmi dlouhé řádky (10k).
`-fimplicit-none`	AUtomatické `IMPLICIT NONE`. Lze přebít explicitními `IMPLICIT` příkazy.
`-fintrinsic-extensions`	Zapne vestavěné funkce g95 i v `-std=` módu.
`-fintrinsic-extensions=`	Totéž, ale selektivní (jména oddělena čárkami).
`-fmod=`*directory*	Umisťuje .mod soubory do adresáře *directory*.
`-fmodule-private`	Nstaví defaultní práva entit modulů na `PRIVATE`.
`-fmultiple-save`	Povolí vícenásobnou specifikaci atributu `SAVE`.
`-fone-error`	Zastaví překlad po první chybě.
`-ftr15581`	Zapne TR15581 rozšíření alokovatelných polí i v módech `-std=F` a `-std=f95`.
`-std=F`	Drží se standardu F. See `http://www.fortran.com/F`.
`-std=f2003`	Drží se standardu Fortran 2003 (implementované podmožiny).
`-std=f95`	Drží se standardu Fortran 95.
`-i4`	Defaultní integer kind=4 (32 bitů).
`-i8`	Defaultní integer kind=8 (64 bitů).
`-r8`	Defaultní real kind=8 (dvojitá přesnost).
`-d8`	Zapne `-i8` a `-r8`.

Volby generování kódu

`-fbounds-check`	Kontroluje meze polí a řetězců za běhu.
`-fcase-upper`	Všechny veřejné symboly v uppercase.
`-fleading-underscore`	Přidá podtržítko před veřejná jména.
`-fonetrip`	Proveď `DO`-cykly aspoň jednou. (Pro nechodící `FORTRAN` 66).
`-fpack-derived`	Seskládat struktury co nejkompaktněji. Šetří paměť, ale může zpomalovat program.
`-fqkind=`*n*	Nastaví real kind pro konstanty s exponentem 'q' na *n*.
`-fsecond-underscore`	Přidá další podtržítko za jména s podtržítkem (default). Použijte `-fno-second-underscore` pro potlačení.
`-fshort-circuit`	Zkrácené vyhodnocování a la C operátorů `.AND.` a `.OR.`
`-fsloppy-char`	Povoluje přiřazení neznakových dat do znakových proměnných a porovnávání INTEGERových a CHARACTERových proměnných.
`-fstatic`	Alokuje lokální proměnné ze statické paměti kdekoli to jde. Neplést s volbou `-static`.
`-ftrace=`	`-ftrace=frame` vkládá kód pro zpětné trasování chybových ukončení programu, který se tím zpomalí. `-ftrace=full` navíc umožní vypsat číslo řádku aritmetických výjimek (ještě pomalejší). Defaultní je `-ftrace=none`.
`-funderscoring`	Připojí podtržítka za globální jména. Tahle volba je defaultně zapnutá, vypíná se přes `-fno-underscoring`.
`-max-frame-size=`*n*	Mez pro velikost zásobníkového rámce, než se začnou pole alokovat z heapu. *pozn.aut.: default je co?*)
`-finteger=`*n*	Inicializuje skalární celočíselné proměnné na *n*.
`-flogical=`*value*	Inicializuje skalární logické proměnné. *value* je smí být `none`, `true` nebo `false`.
`-freal=`*value*	Inicializuje skalární reálné a komplexní proměnné. *value* smí být `none`, `zero`, `nan`, `inf`, `+inf` nebo `-inf`.
`-fpointer=`*value*	Inicializuje ukazatele. *value* smí být `none`, `null` nebo `invalid`.

`-fround=`*value* Kontroluje zaokrouhlování při překladu. *value* smí být `nearest`, `plus`, `minus` nebo `zero`. Default je zaokrouhlování k bližší hodnotě, `plus` je zaokrouhlování ke kladnému nekonečnu, `minus` k zápornému, `zero` k nule.

`-fzero` Inicializuj numerické typy na nulu, logické proměnné na false a ukazatele na `null`. Výše uvedené inicializační volby tuto přebíjejí.

Adresářové volby

`-I` *directory* Přidá adresář *directory* k prohledávaným cestám pro soubory include a moduly.

`-L`*directory* Přidá adresář *directory* k cestám pro knihovny.

`-fmod=`*directory* Umisťuje .mod soubory do adresáře *directory*.

Environment Variables

Běhové prostředí g95 poskytuje hodně možností pro změnu chování programu po spuštění, a to pomocí proměnných prostředí. Spuštění programu přeloženého g95 s volbou `--g95` vypíše všechny tyto volby na standardní výstup. Hodnoty všech proměnných prostředí jsou vždycky řetězce, ale mohou být interpretovány jako čísla. Jen první znak booleovské proměnné se kontroluje, a musí být jeden z '`t`', '`f`', '`y`', '`n`', '`1`' nebo '`0`' (mohou být velká). Je-li hodnota špatná, nedočkáte se žádného varování a použije se defaultní nastavení. Pro informaci o proměnných prostředí GCC používaných g95, jako je `LIBRARY_PATH`, viz dokumentace ke GCC.

`G95_STDIN_UNIT`	Integer	Číslo jednotky předpřipojené ke standardnímu vstupu. Žádná, je-li negativní, default je 5.
`G95_STDOUT_UNIT`	Integer	Číslo jednotky předpřipojené ke standardnímu výstupu. Žádná, je-li negativní, default je 6.
`G95_STDERR_UNIT`	Integer	Číslo jednotky předpřipojené ke standardnímu chybovému výstupu. Žádná, je-li negativní, default je 0.
`G95_USE_STDERR`	Boolean	Výstup z knihoven na stderr místo stdout. Default je Yes.
`G95_ENDIAN`	String	Endian pro neformátované I/O. Hodnoty jsou `BIG`, `LITTLE` nebo `NATIVE`. Default je `NATIVE`.
`G95_CR`	Boolean	Používat CR zanky pro konce záznamů ve formátovaném výstupu. Default je TRUE na Windows (ale ne Cygwin), FALSE jinde.
`G95_INPUT_CR`	Boolean	CRLF na vstupu brát jako LF - konec záznamu. Default je TRUE.
`G95_IGNORE_ENDFILE`	Boolean	Ignoruje čtení za koncem souboru v sekvenčním módu. Default je FALSE.
`G95_TMPDIR`	String	Adresář pro scratch soubory. Není-li nastaveno, užije se proměnná `TMP`. Není-li nastavena `TMP`, použije se `/var/tmp`.
`G95_UNBUFFERED_ALL`	Boolean	Je-li TRUE, veškerý výstup se nebufferuje. Zpomaluje menší zápisy, ale může být užitečné, je-li třeba data okamžitě zapisovat. Default je FALSE.
`G95_SHOW_LOCUS`	Boolean	Je-li TRUE, vypisuje jméno souboru a číslo řádku běhové chyby. Default je TRUE.
`G95_STOP_CODE`	Boolean	Je-li TRUE, kódy STOP se předávají systému. Default TRUE.
`G95_OPTIONAL_PLUS`	Boolean	Vypisuje plus před kladná čísla. Default je FALSE.
`G95_DEFAULT_RECL`	Integer	Defaultní maximální délka záznamu pro sekvenční soubory. Nejužitečnější pro kontrolu délky řádků předpřipojených jednotek. Default je 50000000.
`G95_LIST_SEPARATOR`	String	Oddělovač pro seznamem řízený výstup. Může obsahovat libovolný počet mezer a nejvýš jednu čárku. Default je jediná mezera.
`G95_LIST_EXP`	Integer	Nejvyšší mocnina desítky, pro niž se ještě nepoužije E formát. Default 6.

G95_COMMA	Boolean	Použije čárku jako desetinný oddělovač v I/O. Default FALSE.
G95_EXPAND_UNPRINTABLE	Boolean	Tiskne jinak netisknutelné znaky ve formátovaném výstupu přes \-sekvence. Default FALSE.
G95_QUIET	Boolean	Potlačí znaky pípnutí (\a) ve formátovaném výstupu. Default FALSE.
G95_SYSTEM_CLOCK	Integer	Počet tiků za sekundu, které používá vestavěná funkce SYSTEM_CLOCK(). Nula hodiny vypíná. Default je 100000.
G95_SEED_RNG	Boolean	Je-li TRUE, automaticky provádí random_seed při spuštění programu. Default FALSE.
G95_MINUS_ZERO	Boolean	Je-li TRUE, tiskne nulu vždy bez znaménka. Tradiční ale ne standardní. Default FALSE.
G95_ABORT	Boolean	Je-li TRUE, udělá výpis jádra při abnormálním ukončení programu. Užitečné pro lokalizaci problémů. Default FALSE.
G95_MEM_INIT	String	Kontroluje inicializaci alokované paměti. Default je NONE pro žádnou inicializaci (rychlejší), NAN pro Not-a-Number (v hexu 0x00f95) nebo vybraná hexadecimální hodnota.
G95_MEM_SEGMENTS	Integer	Maximální počet stále alokovaných paměťových segmentů zobrazených při ukončení programu. 0 znamená žádné, méně než 0 znamená všechny. Default je 25.
G95_MEM_MAXALLOC	Boolean	Je-li TRUE, ukazuje se maximální počet bytů alokovaných za běhu programu. Default FALSE.
G95_MEM_MXFAST	Integer	Maximální velikost alokací zpracovávaných přes fastbins. Fastbins jsou rychlejší al snáze se fragmentují. Default je 64 bytů.
G95_MEM_TRIM_THRESHOLD	Integer	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	Integer	Množství paměti navíc při alokaci od systému. Může zrychlit budoucí alokace. Default 0.
G95_SIGHUP	String	Chování programu při signálu SIGHUP: IGNORE, ABORT, DUMP nebo DUMP-QUIT. Default ABORT. Jen na Unixech.
G95_SIGINT	String	Totéž při SIGINT.
G95_SIGQUIT	String	Totéž při SIGQUIT.
G95_CHECKPOINT	Integer	Na Linuxu x86, počet sekund mezi checkpointy (výpisy corefile), nula znamená bez výpisů.
G95_CHECKPOINT_MSG	Boolean	Je-li TRUE, ohlásí checkpoint procesu na stderr. Default TRUE.
G95_FPU_ROUND	String	Nastavuje režim zaokrouhlování v plovoucí čárce. Může být NEAREST, UP, DOWN, ZERO. Default je NEAREST.
G95_FPU_PRECISION	String	Přesnost mezivýsledků. Může být 24, 53 a 64. Default 64. Jen na x86 a kompatibilních.
G95_FPU_DENORMAL	Boolean	Vyvolá výjimku plovoucí čárky při výskytu denormalizované hodnoty. Default FALSE.
G95_FPU_INVALID	Boolean	Vyvolá výjimku plovoucí čárky při neplatné operaci. Default FALSE.
G95_FPU_ZERODIV	Boolean	Vyvolá výjimku plovoucí čárky při dělení nulou. Default FALSE.
G95_FPU_OVERFLOW	Boolean	Vyvolá výjimku plovoucí čárky při přetečení. Default FALSE.
G95_FPU_UNDERFLOW	Boolean	Vyvolá výjimku plovoucí čárky při podtečení. Default FALSE.
G95_FPU_INEXACT	Boolean	Vyvolá výjimku plovoucí čárky při ztrátě přesnosti. Default FALSE.
G95_FPU_EXCEPTIONS	Boolean	Zda mají být maskované výjimky plovoucí čárky ukázány na konci programu. Default FALSE.
G95_UNIT_x	String	Přenastaví defaultní jméno pro I/O jednotku x. Default je fort.x
G95_UNBUFFERED_x	Boolean	Je-li TRUE, I/O jednotka x není bufferována. Default FALSE.

Běhové chybové kódy

Spuštění programu přeloženého g95 s volbou –g95 vypíše tento seznam chybových kódů na standardní výstup.

-2	Konec záznamu
-1	Konec souboru
0	Úspěšné ukončení
	Kódy operačního systému (1 - 199)
200	Conflicting statement options
201	Bad statement option
202	Missing statement option
203	Soubor již otevřen v jiné jednotce
204	Nepřipojená jednotka
205	Chyba FORMAT
206	Nesprávná ACTION
207	Čtení za záznam ENDFILE
208	Špatná hodnota během čtení
209	Numerické přetečení při čtení
210	Nedostatek paměti
211	Pole již je alokováno
212	Dealokace špatného pointeru
214	Neplatný záznam v neformátovaném sekvenčním I/O
215	Čtení více dat než je velikost záznamu (RECL)
216	Zápis více dat než je velikost záznamu (RECL)

Věci z Fortranu 2003

G95 implementuje několik částí Fortranu 2003. Viz diskuse o všech novinkách Fortranu 2003 na: `http://www.kcl.ac.uk/kis/support/cit/fortran/john_reid_new_2003.pdf`.

- Jsou k dispozici následující vestavěné procedury: `COMMAND_ARGUMENT_COUNT()`, `GET_COMMAND_ARGUMENT()`, `GET_COMMAND()` a `GET_ENVIRONMENT_VARIABLE()`
- Reálné indexy DO cyklů nejsou povoleny.
- Lze užívat hranaté závorky `[` a `]` coby alternativu k `(/` a `/)` pro konstruktory polí.
- TR 15581 - rozšíření alokovatelných entit. Umožňuje specifikovat atribut `ALLOCATABLE` pro formální argumenty procedur, návratové hodnoty funkcí a komponenty struktur.
- Stream I/O - specifikace FORM='STREAM' umožňuje Fortranskému programu číst a zapisovat binární soubory bez používání záznamů. Clive Page napsal pojednání o této schopnosti: `http://www.star.le.ac.uk/~cgp/streamIO.html`.
- Příkaz `IMPORT`. Lze užít v těle INTERFACE pro zpřístupnění entit z okolí (modulu, procedury).
- Evropská konvence pro reálná čísla – volba `DECIMAL='COMMA'` v příkazech `OPEN`, `READ` a `WRITE` podporuje reálná čísla s desetinnou čárkou místo tečky.
- `MIN()` a `MAX()` fungují i s řetězci a znaky.
- Atribut `VALUE` umožňuje specifikovat předávání parametru hodnotou.
- Konstruktory struktur z F2003.
- Procedurální ukazatele z F2003.
- Interoperabilita s C - konstrukce `BIND(C)`, modul `ISO_C_BINDING`.

Vícejazyčné programování

Ačkoli g95 produkuje samostatné programy, občas je potřeba spolupráce si jinými jazyky, nejčastěji C. První potíž s kombinací více jazyků jsou jména veřejných symbolů. G95 se drží konvence f2c: připojuje za veřejná jména podtržítko, dvě podtržítka za jména obsahující podtržítko. Volbami `-fno-second-underscore` a `-fno-underscoring` lze přimět g95 k vyprodukování jmen kompatibilních s vaším C překladačem. Můžete použít utilitu `nm` na objektové `.o` soubory produkované oběma překladači ke kontrole vyprodukovaných jmen.

G95 produkuje veřejná jména v malých písmenech, pokud mu nezadáte volbu `-fupper-case`, v kterémžto případě použije velká písmena. Jména entit v modulech jsou reprezentovány jako *module-name*`_MP_`*entity-name*.

Při vícejazyčném programování lze rozlišit dva hlavní případy: Volání C procedur z Fortranu, a volání Fortranských procedur z C. První případ nic speciálního nepotřebuje (pozor, neplatí pro C++). Ve druhém případě, volání Fortranu z C, budou Fortranské podprogramy někdy volat knihovní funkce, které očekávají nějak inicializovanou haldu a jiné věci. Běhové prostředí g95 lze inicializovat z C voláním `g95_runtime_start()` a finalizovat pomocí `g95_runtime_stop()`. Prototyp `g95_runtime_start()` je:

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

Není-li přístup k argumentům příkazové řádky (např. jde o knihovnu), nebo nejsou-li potřeba, lze předat `argc=0` a `argv=NULL`. Na OSX, použijte `-lSystemStubs` při vícejazyčném programování.

F2003 spoupráci s C velmi zjednodušuje. Atribut `BIND(C)` umožňuje kontrolovat jména Fortranských symbolů tak, aby se na ně bylo možné snáze odkazovat z C (nebo jiných jazyků). Například:

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

Tento tvar vytvoří symbol jménem `foo` bez přidaných podtržítek. Všechna písmena jsou malá. Podobný tvar je:

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

Tím se symbol pojmenuje `Foo1`. Unitř Fortranu se na podprogram stále odkazujeme jako na `foo`, `FOO` atd.

Programy v C předávají argumenty hodnotoum zatímco Fortran (normálně) odkazem. F2003 atribut `VALUE` specifikuje argumenty předávané hodnotou. Příklad:

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

Podprogram takto definovaný lze z Fortranu stále normálně volat s tím omezením, že formální argumenty už nejsou asociovány s aktuálními a změna formálního argumentu se na aktuálním neprojeví.

Globální proměnné lze zpřístupnit podobně. Následující podprogram vypíše hodnotu proměnné `VAR`, která by jinak byla z Fortranu nepřístupná.

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

Zatímco fortran rozlišuje různé druhy (kinds) základních typů, C definuje vše jak různé typy. Pro zajištení vzájemné korespondence typů obsahuje vestavěný modul `ISO_C_BINDING` následující druhové konstanty (`PARAMETER`):

`c_int`	Integer kind pro C `int`
`c_short`	Integer kind pro C `short`
`c_long`	Integer kind pro C `long`
`c_long_long`	Integer kind pro C `long long`
`c_signed_char`	Integer kind pro C `char`
`c_size_t`	Integer kind pro C `size_t`
`c_intptr_t`	Integer kind of the same size as C pointers
`c_float`	Real kind pro C `float`
`c_double`	Real kind pro C `double`

V modulu `ISO_C_BINDING` je i spousta dalších věcí. Příklad použití:

```
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
    ...
```

Použití generátoru pseudonáhodných čísel

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

Naplní `REAL` proměnnou `harvest` (skalár nebo pole) pseudonáhodným čísly , $0 \leq$ `harvest` < 1.
Nastavení seedu generátoru:

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

`sz` je minimální počet integerů potřebných pro seed, g95 stačí čtyři. Argument `pt` je pole integerů velikosti `n1` $\geq$ `sz`, obsahující uživatelské hodnoty seedu. Argument `gt` je pole integerů velikosti `n2` $\geq$ `sz`, do nějž se uloží dosavadní seed.

Volání `RANDOM_SEED()` bez argumentů inicializuje seed na základě aktuálního času - tím se zajistí generování různých posloupností pseudonáhodných čísel pokaždé, když je program znovu spuštěn. Totéž se provede automaticky, je-li nastavena proměnná prostředí `G95_SEED_RNG` na TRUE. V opačném případě, `RANDOM_NUMBER()` vždy generuje stejnou sekvenci.

Použitý generátor je xor-shift generátor, jehož autorem je George Marsaglia.

P reddefinovaná makra preprocesoru

Vždy jsou definována makra:

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

Podmíněná makra jsou:

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

Schopnost Corefile Resume

Na x86 Linuxových systémech lze provádění programu přeloženého g95 pozastavit a obnovit. Přeruší-li se program signálem QUIT, obvykle generovaným stiskem Ctrl-zpětné lomítko, program vytvoří v aktuálním adresáři spustitelný soubor jménem `dump` (ale nepřeruší běh). Spuštění tohoto souboru kdykoli později po skončení nebo přerušení programu, obnoví provádění vašeho programu v místě, kde byl checkpoint pořízen. Ilustruje to následující sezení:

```
andy@fulcrum:~/g95/g95 % cat tst.f90
  b = 0.0
  do i=1, 10
      do j=1, 3000000
          call random_number(a)
          a = 2.0*a - 1.0
          b = b + sin(sin(sin(a)))
      enddo
      print *, i, b
  enddo
  end
andy@fulcrum:~/g95/g95 % g95 tst.f90
andy@fulcrum:~/g95/g95 % a.out
 1 70.01749
 2 830.63153
 3 987.717
 4 316.48703
 5 -426.53815
```

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

Otevřené soubory musí být přítomny na stejných místech jako v původním procesu. Při vícejazyčném programování schopnost nemusí fungovat. Nejdůležitější použití je zachování běhu přes reboot nebo odhlášení, lze použít i k překonání limitů na behový čas procesů, nebo i přenos běžícího programu na jiný stroj. Automatické generování checkpointů lze zapnout nastavením proměnné prostředí `G95_CHECKPOINT` na počet sekund mezi výpisy. Nula znamená bez výpisů. Nový výpis přepíše starý.

Chytrý překlad

Uvažujme modul `foo` jehož zdroj je v souboru `foo.f95`. Lze rozlišit dva druhy změn ve `foo.f95`:

1. Změny které mění vnější vzhled `foo`, např. změna typu nebo rozhraní procedury;
2. Vnitřní změny, ovlivňující jen implementaci, např. oprava chyby v kódu.

Oba druhy změn obecně změní objektový kód `foo.o`, ale jen první druh opravdu změní foo.mod. Když g95 znovu překládá modul, detekuje zda soubor `.mod` potřebuje update, a pokud jde o změnu typu 2, ponechá starý soubor `.mod` nezměněný (včetně data).

Tato schopnost g95 brání zbytečným kompilačním kaskádám při sestavování velkých programů. Zavisí-li mnoho různých zdrojáků na `foo.mod`, buď přímo (kvůli `USE FOO`) nebo nepřímo (užitím modulu který používá `foo`, nebo užitím modulu který používá modul který používá `foo`, atd). Změna typu 1 ve `foo.f95` způsobí rekompilaci všech závislých zdrojových souborů; naštěstí takové změny nejsou nejčastější (alespoň je-li interface pečlivě navrženo předem). Běžnější změny typu 2 způsobí jen rekompilaci `foo.f95` samotného, načež lze rovnou sestavit program s novým `foo.o`.

G95 Rozšířené vestavěné funkce

ACCESS

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

Kontroluje zda je soubor filename přístupný ve specifikovaném módu, kde mód je jedno nebo více písmen `rwxRWX`. Vrátí nulu je-li to tak, nenulovou hodnotu, je-li něco špatně.

ALGAMA

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

Počítá přirozený logaritmus $\Gamma(x)$. `ALGAMA` je elementální funkce použitelná na jakýkoliv reálný typ.

BESJ0

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

Počítá Besselovu funkci nultého řádu prvního druhu. Elementální funkce.

BESJ1

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

Počítá Besselovu funkci prvního řádu prvního druhu. Elementální funkce.

BESJN

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

Počítá Besselovu funkci n-tého řádu prvního druhu. Elementální funkce.

BESY0

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

Počítá Besselovu funkci nultého řádu druhého druhu. Elementální funkce.

BESY1

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

Počítá Besselovu funkci prvního řádu druhého druhu. Elementální funkce.

BESYN

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

Počítá Besselovu funkci n-tého řádu druhého druhu. Elementální funkce.

CHMOD

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

Změní unixová práva k souboru. Vrací nenulovou hodnotu při chybě.

DBESJ0

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

viz BESJ0.

DBESJ1

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

viz BESJ1.

DBESJN

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

viz BESJN.

DBESY0

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

viz BESY0.

DBESY1

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

viz BESY1.

DBESYN

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

viz BESYN.

DCMPLX

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

Double precision `CMPLX`, `x` a `y` mohou být libovolného typu a druhu.

DERF

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

viz ERF.

DERFC

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

viz ERFC.

DFLOAT

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

Alias pro `DBLE`.

DGAMMA

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

Viz GAMMA.

DLGAMA

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

viz ALGAMA.

DREAL

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

Alias pro `DBLE`.

DTIME

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

viz ETIME.

ERF

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

Počítá chybovou funkci `x`. Elementální funkce.

ERFC

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

Počítá doplňkovou chybovou funkci `x`. Elementální funkce.

ETIME

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

Nastaví `tarray(1)` na počet uplynulých sekund uživatelského času v aktuálním procesu. `tarray(2)` nastaví na počet uplynulých sekund systémového času v aktuálním procesu. Vrací součet těchto časů.

FNUM

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

Vrací deskriptor odpovídající Fortranské I/O jednotce `unit`. Vrací -1 není-li jednotka připojena.

FSTAT

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

Získá údaje o souboru otevřeném na I/O jednotce `unit` a uloží je do pole `sarray(1:13)`. Hodnoty jsou extrhovány ze struktury stat používané libc funkcí fstat(): `sarray(1)` Číslo zařízení, `sarray(2)` číslo Inode, `sarray(3)` mód souboru, `sarray(4)` počet linků, `sarray(5)` uid vlastníka, `sarray(6)` gid vlastníka, `sarray(7)` typ zařízení, `sarray(8)` velikost souboru. `sarray(9)` čas přístupu, `sarray(10)` čas modifikace, `sarray(11)` čas změny, `sarray(12)` velikost bloku, `sarray(13)` alokované bloky.

FDATE

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

Vrací aktuální datum a čas jako Day Mon dd hh:mm:ss yyyy.

FTELL

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

Vrátí aktuální offset souboru na jednotce `unit` nebo -1 není-li jednotka připojena.

GAMMA

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

Počítá $\Gamma(x)$. `GAMMA` je generická funkce.

GETCWD

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

Uloží aktuální adresář do `name`. Vrací nenulovou hodnotu v případě chyby.

GETGID

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

Vrací id skupiny aktuálního procesu.

GETPID

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

Vrací id aktuálního procesu.

GETUID

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

Vrací id uživatele.

HOSTNM

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

Nastaví `name` name na systémové hostitelské jméno. Vrací nenulovou hodnotu v případě chyby.

IARGC

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

Vrací počet argumentů příkazové řádky (mimo program samotný).

ISATTY

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

Vrací `.true.` je-li I/O jednotka `unit` připojena k terminálu.

ISNAN

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

Vrací `.true.` je-li `x` Not-a-Number (NaN). Elementální funkce.

LINK

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

Vyrobí tvrdý odkaz `path1` na `path2`.

LNBLNK

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

Alias pro standardní funkci `len_trim`.

LSTAT

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

Je-li `file` symbolický odkaz vrací data o odkazu samotném. Viz `FSTAT()`. Vrací nenulovou hodnotu v případě chyby.

RAND

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

Varcí pseudonáh. číslo s rovnom. rozdělením v intervalu $0 \leq$ `rand` < 1. Je-li `x` rovno 0, vrací se další číslov sekvenci. Je-li `x` rovno 1, generátor se restartuje voláním `srand(0)`. Má-li `x` jinou hodnotu, je použita jako nový seed pro srand().

SECNDS

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

Vrací místní čas v sekundách od půlnoci minus `t`.

SIGNAL

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

Interface k unixové funkci `signal`. Vrací nenulovou hodnotu v případě chyby.

SIZEOF

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

Argument `object` je výraz, proměnná nebo typ. Vrací velikost `object` v bytech.

STAT

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

viz FSTAT.

SYSTEM

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

Spustí externí příkaz `cmd`. Vrátí jeho návratový kód.

TIME

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

Vrací aktuální čas zakódovaný v celém čísle jako u UNIXové funkce `time`.

UNLINK

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

Odlinkuje (smaže) soubor `file`. Vrací nenulovou hodnotu v případě chyby.

%VAL()
Aplikována na aktuální argument při volání funkce, způsobí jeho předání hodnotou. Tato pseudofunkce není doporučena, a je zahrnuta jen kvůli kompatibilitě. F2003 atribut `VALUE` je standardní mechanismus.

%REF()
Totéž, jen zajišťuje předání odkazem.

G95 rozšířené vnitřní subrutiny

ABORT

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

Ukončí program s výpisem jádra zasláním signálu SIGABORT sobě (Unix).

CHDIR

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

Naství aktuální adresář na `dir`.

DTIME

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

viz funkce ETIME.

ETIME

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

viz funkce ETIME.

EXIT

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

Ukončí program s výstupním kódem `code` po zavření otevřených I/O jednotek. Generická subrutina.

FDATE

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

viz funkce FDATE.

FLUSH

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

Spláchne I/O jednotku `unit` otevřenou pro výstup.

FSTAT

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

viz funkce FSTAT.

GETARG

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

Nastaví proměnnou `value` na `pos`-tý arhument příkazové řádky.

GETENV

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

Nastaví proměnnou `value` na hodnotu proměnné prostředí `variable`.

GETLOG

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

Vrací přihlašovací jméno procesu v proměnné `name`.

IDATE

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

Nastaví `m` na aktuální měsíc, `d` na den v měsíci a `y` na aktuální rok. Tahle subrutine není příliš přenositelná V nových kódech použijte standardní subrutinu `DATE_AND_TIME`.

LSTAT

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

Viz funkce LSTAT.

RENAME

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

Přejmenuje soubor `path1` na `path2`. Je-li přítomen argument `status`, je nastaven na nenulovou hodnotu v případě chyby.

SIGNAL

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

Viz funkce SIGNAL.

SLEEP

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

Způsobí čekání procesu na `seconds` sekund.

SRAND

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

Reinicializuje generátor pseudonáhodných čísel. Viz C funkce `srand()`.

STAT

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

Viz funkce FSTAT.

SYSTEM

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

Viz funkce SYSTEM.

UNLINK

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

Viz funkce UNLINK.