

NAME

fcpp – Fortran C Preprocessor

SYNOPSIS

fcpp [-D*name*[=*value*]] [-U*name*] [-F] [-C] [-I*directory*] [-V] [*input-file* [*output-file*]]

DESCRIPTION

fcpp is a modified C preprocessor designed to be used on Fortran code. It is a modified version of **ccpp**, the C Compatible Compiler Preprocessor written by Paul Rubin and copyrighted by the Free Software Foundation. This preprocessor provides all the functionality of the old style C preprocessor as well as *#elif* and the macros; `__LINE__`, `__DATE__`, `__FILE__`, and `__TIME__`. In addition, it contains additional features for Fortran code, and it has been ported to the VAX/VMS operating system in addition to other Unix platforms.

Command line options

- F** The input file is Fortran. The preprocessor variable, `LANGUAGE_FORTRAN`, is automatically defined.
- D*name*[=*value*]** The preprocessor variable, *name*, is set to *value* if *value* is specified, and is set to 1 if no *value* is specified.
- U*name*** The preprocessor variable, *name*, is undefined.
- C** C comments are copied to the output file.
- V** The version number of **fcpp** is written to standard error.
- input-file* The file to be processed. If specified as a hyphen ("-") or if omitted, then standard input is used.
- output-file* The output of processing. If specified as a hyphen ("-") or if omitted, then standard output is used.

Directives

The following directives are processed: `#define`, `#undef`, `#include`, `#ifdef`, `#if`, `#ifndef`, `#elif`, `#else`, `#endif`, `#pragma`, and `#line`. Their meaning is the same as in **ccpp** except for the special processing done in VMS for `#include`.

Differences from ccpp

fcpp has been modified as follows:

1. A Fortran mode switch (**-F**) has been added that controls the use of Fortran features.
2. Output lines are continued if they exceed 72 characters when Fortran mode is on.
3. A number of changes have been made to port **fcpp** to VMS. In particular, the I/O in **fcpp** has been rewritten to work line by line rather than with single read's and write's. In version 2.4 of VAXC, this was a necessity because the I/O system failed when large (>32K) read's and write's were attempted.
4. Command option processing is case insensitive on VMS.
5. On VAX/VMS, the first "/" in an included file name is converted to ":" to allow the use of logical names.
6. On VAX/VMS, `SY$LIBRARY:` is searched for included files specified with angle brackets, "<>".
7. In Fortran mode, double quotes (") are ignored. Substitutions within double quotes are allowed. However, single quotes are interpreted as string delimiters within Fortran statements, and no substitutions are allowed in the string. Single quotes are not specially processed within Fortran comments, but there is a difficult bug which prevents **fcpp** from identifying a comment in the first line of a file. Therefore, an unmatched single quote in the first line of a file which is a comment will result in no preprocessing.

8. The VAX version of the program has the preprocessor variable, *vax*, defined. The VMS version has *vax*, *VAX*, *vms*, and *VMS* defined. The Unix version has *unix* defined.
9. If an error occurs handling an `#include` directive on VMS, the directive is written to the output file. This gives the VAX C compiler another crack at it.
10. On VMS, no `#line` directive is written out after an `#include` directive.

DIAGNOSTICS

Self-explanatory (hopefully!). Syntax errors give the line number for the bad line.

SEE ALSO

`cpp(1)`, `cc(1)`.

FILES**CAVEATS**

Don't put a single quote in a comment found in the first line of Fortran file.

ORIGIN

Paul Rubin and Free Software Foundation.
Modifications by Robert E. Brucoleri
Bristol-Myers Squibb Pharmaceutical Research Institute

COPYLEFT

Copyright (C) 1986, Free Software Foundation, Inc.
See the beginning of the source file, `cccp.c`, for the full text of the License Agreement covering use and copying of this program.

NAME

mkproto – Make prototypes for functions

SYNOPSIS

mkproto

[-n] [-s] [-p] [-E] [-i name] [-l int] [file] ...

DESCRIPTION

mkproto takes as input one or more C source code files, and produces as output (on the standard output stream) a list of function prototypes (a la ANSI) for the external functions defined in the given source files. This output, redirected to a file, is suitable for `#include`'ing in a C source file.

The function definitions in the original source may be either "old-style" (in which case appropriate prototypes are generated for the functions) or "new-style" (in which the definition includes a prototype already).

A **-n** option causes the line number where each function was defined to be prepended to the prototype declaration as a comment.

A **-s** option causes prototypes to be generated for functions declared "static" as well as extern functions.

A **-p** option causes the prototypes emitted to be only readable by ANSI compilers. Normally, the prototypes are "macro-ized" so that compilers with `__STDC__` not defined don't see them.

A **-E** option causes all preprocessor statements in the to be emitted to the prototype file. If you use this option, it is important that all your header files be idempotent, namely, that they can be reincluded multiple times with the same effect as a single inclusion.

The **-i name** option is used to make the emitted prototype file idempotent. The **name** is used to conditionally compile the prototype file as a whole, and therefore, the prototypes will only be seen once by the compiler. Try this option out to see the effect.

The **-l int** option controls the width of lines output by **mkproto**. After each variable in a functions prototype is output, **mkproto** checks to see if the current line length exceeds the internal variable, `breakafter`. If it does, then a newline followed by appropriate indentation is output. Thus, this variable doesn't specify a hard line width, it specifies an approximate right margin. Only positive values are permitted for this option. A value of 1 results in each variable having its own line.

If files are specified on the command line, then a comment specifying the file of origin is emitted before the prototypes constructed from that file. If no files are given, then no comments are emitted and the C source code is taken from the standard input stream.

BUGS

1) **mkproto** is easily confused by complicated declarations, such as

```
int ((*signal)())() { ...
```

or

```
struct foo { int x, y; } foofunc() { ...
```

2) Float types are not properly promoted in old style definitions, i.e.

```
int test(f) float f; { ...
```

should (because of the default type conversion rules) have prototype

```
int test(double f);
```

rather than the incorrect

```
int test(float f);
```

generated by **mkproto**.

3) Some programs may need to be run through the preprocessor before being run through **mkproto**. The **-n** option is unlikely to work as desired on the output of a preprocessor.

4) typedef'd types aren't correctly promoted, e.g. for

```
typedef schar char; int foo(x) schar x; ...
```

mkproto incorrectly generates the prototype `int foo(schar x)` rather than the (correct) `int foo(int x)`.

5) Functions named "inline" with no explicit type qualifiers are not recognized.

SEE ALSO

`cc(1)`, `lint(1)`

AUTHORS

Eric R. Smith.

Robert E. Brucoleri.

NOTE

There is no warranty for this program (as noted above, it's guaranteed to break sometimes anyways!). **mkproto** is in the public domain.

NAME

wrapgen – Wrapper generator for Fortran C interlanguage calls

SYNOPSIS

wrapgen

[-c] [-f] [-m] [-i *input-file*] [-o *output-file*] [-h *header-file*]

DESCRIPTION

wrapgen generates wrappers for C to Fortran and Fortran to C interlanguage calls. These wrappers are designed to hide the machine dependencies inherent in interlanguage calls. Programs using **wrapgen** can then use C and Fortran in a portable way. **wrapgen** takes care of converting character string parameters and call by value parameters.

Command line options

- c** Write wrappers for procedures written in C
- f** Write wrappers for subprograms written in Fortran
- m** Writes multiple files. A separate file beginning with the prefix **wr_** is written for every wrapper. This feature is intended for building object libraries where each wrapper is isolated into its own module. When this option is specified, the **-o *output-file*** option is ignored.
- h *header-file*** Specifies the header file which must be included into any source file which calls the wrapped functions. This header file is written by **wrapgen**. See below for more information.
- i *input-file*** Specifies the input file of prototypes which describe the functions for which wrappers are to be generated. Defaults to standard input.
- o *output-file*** Specifies the file where the wrapper procedures are written. This is a C language source file that must be compiled and linked into the program using the wrappers.

Usage

wrapgen generates wrappers for either Fortran or C procedures, depending on the choice of the **-c** or **-f** option. In either case, the specifications for the wrappers are done using the same syntax, namely that of C function prototypes. The syntax for the prototypes is as follows:

prototype ::= [data-type] {function-name} (arglist) ;

arglist ::= <empty> | void | arglist1

arglist1 ::= arg | arg , arglist1

arg ::= data-type [*] variable-name

data-type ::= int | float | double | char | void |
F77_INTEGER | F77_REAL | F77_DOUBLE |
F77_LOGICAL | F77_POINTER

Any number of wrappers may be specified in the input file.

There are three different types of variables; character, size, and scalar. Character variables are used to pass character strings, and always specified as pointers to *char*. Size variables are integers or F77_INTEGER's, not declared as pointers, which have the same variable name as the corresponding character string, except that "**_size**" is appended to the end. Size variables are described in more detail below. Scalar variables are integers, logicals, floats, double precision floats, and Fortran pointers. Fortran pointers are variables in

Fortran that are intended to hold addresses used in C code. All can be passed by pointers, and all scalar variables except double precision floats can be passed by value.

The return type of any prototype should be limited to *void*, integers, logicals, single precision floats, and integers big enough to hold pointers (F77_POINTER). Double precision floats are difficult to return, and **wrapgen** does not handle them.

A number of preprocessor variables must be defined for the wrappers to work. See the **Implementation** section below for more information.

Wrappers for functions defined in Fortran

The key transformation that **wrapgen** provides is the conversion of character strings in C usage to that expected by Fortran subroutines. In Fortran, all character strings have a size associated with them. The wrappers generated by **wrapgen** can either calculate this size using *strlen* or it can be provided by the programmer. **wrapgen** will choose based on whether a *_size* variable exists for the character string as specified in the prototype. Such size variables are integers passed by value which have "*_size*" appended to the character string variable name. You must decide when writing the procedure which way a string is being passed, and then this form must be used throughout the code.

wrapgen will also transform parameters passed by value into parameters passed by reference. The wrapper's copy of the parameter will be used to provide storage. The presence or absence of an asterisk ("*") before the variable name in the prototype signifies that the C code expects a parameter to be passed by value.

The header file produced by the **-h** option contains *#define* statements which rename the functions to their wrapper form (the name specified in the prototype plus "*_wrap*"), and they also contain a function prototype for this wrapper so that no parameter promotions will be done. This is important for calling Fortran subroutines which expect float parameters and for which the C call uses a call by value.

Procedure for Fortran defined subprograms

For each subroutine which is called from C, you must do the following:

- 1) Define a function prototype in C which specifies how your C code will call the subroutine. The order of character strings and other parameters must match the Fortran code. You have the option of specifying size variables for each of the character strings; these may be specified in any position, but must be consistently used. The names must correspond to the character strings with the addition of "*_size*" to the name. If you omit a size variable for a character string, then the wrapper will use *strlen* to get a value. In the Fortran code, simply declare these variables as CHARACTER.
- 2) You can also declare *int*, *float*, *F77_REAL*, *F77_INTEGER*, *F77_POINTER*, and *F77_LOGICAL* without a pointer (call by value), and the wrapper will generate a pointer for you.
- 3) You must include the header that is produced by **wrapgen** in any file of C code where any Fortran subroutine is called.
- 4) You must define a "config.h" file for use by the wrapper code which defines the preprocessor variables listed under "**Implementation**". The wrapper code must be compiled and linked with your programs.

Example wrapper specification read by **wrapgen**:

```
void gtrmni (    char *comlyn, int comlyn_size, int *comlenp,
               char *keyst, int *valp, int def);
```

gtrmni is a subprogram which will search the string, *comlyn*, for a keyword, *keyst*, and if found, will convert the following word into an integer and put the result into **valp*. In addition, both the keyword and the following word will be deleted from *comlyn*. If not found, it will copy *def* into **valp*. A sample call to *gtrmni* would look like:

```
#define MXCMSZ 2000
char comlyn[mxcmsz];
int comlen,unit;
gtrmni(comlyn,MXCMSZ,&comlen,"UNIT",&unit,-1);
```

Since *comlyn* is modified, we must have both a maximum length (the size variable in the declaration, *comlyn_size*) and a current length, **comlenp*. Since the keyword, **keyst*, is readonly and would never contain a null character, the use of *strlen* to get the length would make sense. Thus, no size variable appears in the prototype. All other variables are declared as integers. Since *comlen* and *unit* can be changed, they are passed by references. *def* is passed by value since it is a constant. The subroutine call itself is declared *void* because no value is returned.

Wrappers for functions defined in C

Again, the key transformation is for character strings. However, here Fortran provides all the necessary information already; it is up to the programmer to decide if the C functions needs it. The pointer to the character string is always provided, but the size is provided only if a size variable is included in the prototype.

The header file produced here contains *#define* statements which will rename the Fortran call to their *_wrap* forms. Both upper and lower case name conversions are done, but any mixtures of upper and lower case characters in a name are not supported. You must use a C preprocessor like *fcpp* with your Fortran code.

Procedure for C defined procedures

To use C functions from Fortran, you must do the following

- 1) Define C function prototypes that specify the parameters for your C function. All character strings and other parameters must specified in the same order in the definition and in the Fortran call.
- 2) Decide for which character strings you will require the actual lengths as provided by Fortran. Declare these as size variables, and place them wherever you like, although right after the character is recommended.
- 3) Decide on any call by value variables. Any *int*, *float*, *F77_INTEGER*, *F77_REAL*, *F77_LOGICAL* or *F77_POINTER* variable may be declared in the prototype without a pointer indication, and these will be dereferenced before passing to your C function.
- 4) Include the header file produced by **wrapgen** in all Fortran code which calls these C functions. Since the header file will increase the size of the subprogram name in a call, watch out that the limit of 19 continuation lines is not exceeded.
- 5) You must define a "config.h" file for use by the wrapper code which defines the preprocessor variables listed under "Implementation". The wrapper code must be compiled and linked with your programs.

Example wrapper specification read by **wrapgen**:

```
int get_vm (int len, char *callee, int callee_size)
```

get_vm is a subroutine that will return the address of newly allocated memory of length **len*. If the call fails, it will issue an error message containing the string pointed to by *callee* which has a length of *callee_size*. The Fortran call to this routine might look like

```
INTEGER SPACE,GET_VM,LEN
SPACE = GET_VM(LEN,"ALLHP")
```

The *len* argument will be dereferenced when the C function is called.

Implementation

wrapgen is implemented using **yacc** and **lex** for parsing the prototypes. The wrappers are generated using a large number of preprocessor statements so that they can be conditionally compiled for each machine type. A number of preprocessor variables are examined to create the wrappers:

APPEND_FOR	Defined if Fortran entry points have an underscore appended to their names.
vms	Defined for vms systems

ST_BY_EXTRA_LEN Defined for systems which pass character string lengths by adding extra parameters to the end of a subroutine call.

unicos Defined for Cray Unicos

PROTOTYPES Defined for systems where the C compilers can handle prototypes. The use of wrappers without this variable being defined has not been tested.

Note that all of these variables **MUST** be properly defined; defining **vms** and **APPEND_FOR** will yield erroneous results.

DIAGNOSTICS

Self-explanatory (hopefully!). Syntax errors give the line number for the erroneous prototype, but the messages only localize to the nearest semicolon.

SEE ALSO

Congen, fcpp.

FILES

CAVEATS

Only support for VAX/VMS, Iris 4D, Convex, IBM RS/6000, HP 700 series workstations running HP/UX, and Crays running Unicos are currently implemented. DECstations and Sun Sparc based machines should work with the current preprocessor variables.

When working with **wrapgen**, keep in mind that it performs two different functions. The fact that the prototypes are written using C is very confusing.

AUTHOR

Robert E. Bruccoleri
Bristol-Myers Squibb Pharmaceutical Research Institute
P.O. Box 4000
Princeton, NJ 08543-4000 USA
bruc@bms.com

BUGS/COMMENTS

If you use this program, I would really appreciate your comments about it, both positive and negative. Please send me mail.

COPYRIGHT

CONGEN™ Conformational Search and Molecular Modeling System

Copyright © 1987-1988 Robert E. Bruccoleri

Copyright (c) 1990-1997 Bristol-Myers Squibb Company

This software and related documentation is being provided by the copyright holders under the following license. By obtaining, using and/or copying this software, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice and this permission notice and warranty disclaimer appear in all copies.

ROBERT E. BRUCCOLERI AND BRISTOL-MYERS SQUIBB COMPANY DISCLAIMS, AND THE USER WAIVES, ALL REPRESENTATIONS AND WARRANTIES, EXPRESS OR IMPLIED, WITH REGARD TO THIS SOFTWARE AND ITS RELATED DOCUMENTATION, INCLUDING WITHOUT LIMITATION ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR USE OR PURPOSE. ROBERT E. BRUCCOLERI AND BRISTOL-MYERS SQUIBB COMPANY MAKES NO REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, AS TO WHETHER THE USE OF THIS SOFTWARE AND ITS RELATED DOCUMENTATION INFRINGES ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF ANY OTHER PARTY. IN NO EVENT SHALL ROBERT E. BRUCCOLERI OR BRISTOL-MYERS SQUIBB

COMPANY BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, OR ANY OTHER DAMAGES OF ANY NATURE WHATSOEVER, THAT MAY BE INCURRED BY THE USER OR ANY OTHER PARTY ARISING OUT OF OR IN CONNECTION WITH ANY USE OR PERFORMANCE OF THIS SOFTWARE OR ANY USE OF ANY DATA OR RESULTS GENERATED BY THE SOFTWARE, INCLUDING WITHOUT LIMITATION LOSS OF DATA AND LOST PROFITS OR REVENUES, WHETHER OR NOT DR. BRUCCOLERI AND BRISTOL-MYERS SQUIBB COMPANY HAVE BEEN ADVISED OF THE POSSIBILITY OF DAMAGES, AND USER HEREBY WAIVES, RELEASES, AND FOREVER DISCLAIMS ALL DAMAGES, CLAIMS, AND CAUSES OF ACTION IT MAY HAVE AGAINST DR. BRUCCOLERI AND BRISTOL-MYERS SQUIBB COMPANY WITH RESPECT TO ANY LOSSES, DAMAGES, COSTS, EXPENSES, AND LIABILITIES OF ANY NATURE THAT MAY BE INCURRED BY IT ARISING OUT OF OR IN CONNECTION WITH USER'S USE OF THE SOFTWARE AND ITS RELATED DOCUMENTATION. USER SHALL BE SOLELY RESPONSIBLE FOR ALL LOSSES, DAMAGES, COSTS, EXPENSES, AND LIABILITIES OF ANY NATURE INCURRED BY USER RESULTING FROM OR IN CONNECTION WITH USER'S USE OF THE SOFTWARE AND ITS RELATED DOCUMENTATION. THE USER UNDERSTANDS AND ACCEPTS THE ABOVE LIMITATIONS ON DAMAGES AND REMEDIES AS A CONDITION OF OBTAINING USE OF THE SOFTWARE AND RELATED DOCUMENTATION WITHOUT CHARGE.

NAME

fsplit – Split a multi-routine Fortran file into individual files

SYNOPSIS

fsplit [**-e** *efile*] [*files*]

DESCRIPTION

Fsplit takes as input either a file or standard input containing Fortran source code. It attempts to split the input into separate routine files of the form *name.f* where *name* is the name of the program unit (e.g. function, subroutine, block data or program). The name for unnamed block data subprograms has the form *blkdataNNN.f* where NNN is three digits and a file of this name does not already exist. For unnamed main programs the name has the form *mainNNN.f*. If there is an error in classifying a program unit, or if *name.f* already exists, the program unit will be put in a file of the form *zzzNNN.f* where *zzzNNN.f* does not already exist.

-e efiles

Normally each subprogram unit is split into a separate file. When the **-e** option is used, only the specified subprogram units are split into separate files. E.g.:

fsplit -e readit -e doit prog.f

will split readit and doit into separate files.

DIAGNOSTICS

If names specified via the **-e** option are not found, a diagnostic is written to standard error.

HISTORY

Fsplit appeared in 4.2 BSD.

AUTHORS

Asa Romberger and Jerry Berkman

BUGS

Fsplit assumes the subprogram name is on the first noncomment line of the subprogram unit. Nonstandard source formats may confuse **fsplit**.

It is hard to use **-e** for unnamed main programs and block data subprograms since you must predict the created file name.

NAME

ndiffpost – Numerical differences postprocessor

SYNOPSIS

ndiffpost

[-cutoff=*real*] [-angle=*real*] [-sort] [-verbose] [-ignore=*file*] [*input-file*]

DESCRIPTION

ndiffpost is a postprocessing filter for the **diff** program. It goes through all of the difference blocks and reports the maximum absolute or relative differences found for each number in each block. The blocks can be sorted in descending order (using the **-sort** switch). It is very handy for comparing the results of a program after a small change is made to a numerical calculation.

The program expects the *input-file* to be the output of an ordinary **diff** command. Do not use any other options (such as **-c**) with the **diff** program.

An example usage would be

```
diff file1 file2 | ndiffpost -cutoff=1.0e-5 -sort
```

Command line options

- cutoff=*real*** Normally, if a block has no numerical differences, it is not printed, but any numerical differences will result in the block being printed. If this parameter is specified, then any block with maximum differences greater than the **cutoff** will be printed. It is useful for reducing the clutter in an output file. If you want all blocks printed, specify a negative cutoff.
- angle=*real*** If this option is specified, **ndiffpost** will attempt to identify differences in angles (measured in degrees) which bracket 180.0 degrees. For example, if one number is -179.99 and the other is 179.99, then this option can be used to ignore such differences. The *real* number specified as the operand is the cutoff for difference test.
- sort** Blocks will be printed in reverse sorted order of the maximum difference found. This is helpful for scanning big files of differences.
- verbose** Debugging information will be displayed.
- ignore=*file*** If this option is specified, each line in the *file* will be used as a Perl regular expression which is matched against each difference line. If the regular expression matches, the difference line will be ignored. If all difference lines in a block are ignored, then the block itself will also be ignored. This capability is very useful for screening out differences which result from execution time variation or file names changes.

SEE ALSO

perl(1).

IMPLEMENTATION

The script is written in Perl, which was translated from a earlier version written in Awk.

COPYRIGHT

CONGEN(tm) Conformational Search and Molecular Modeling System

Copyright (c) 1987-1988 Robert E. Bruccoleri

Copyright (c) 1990-1997 Bristol-Myers Squibb Company

This software and related documentation is being provided by the copyright holders under the following license. By obtaining, using and/or copying this software, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice and this permission notice and warranty disclaimer appear in all copies.

ROBERT E. BRUCCOLERI AND BRISTOL-MYERS SQUIBB COMPANY DISCLAIMS, AND THE USER WAIVES, ALL REPRESENTATIONS AND WARRANTIES, EXPRESS OR IMPLIED, WITH REGARD TO THIS SOFTWARE AND ITS RELATED DOCUMENTATION, INCLUDING WITHOUT LIMITATION ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR USE OR PURPOSE. ROBERT E. BRUCCOLERI AND BRISTOL-MYERS SQUIBB COMPANY MAKES NO REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, AS TO WHETHER THE USE OF THIS SOFTWARE AND ITS RELATED DOCUMENTATION INFRINGES ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF ANY OTHER PARTY. IN NO EVENT SHALL ROBERT E. BRUCCOLERI OR BRISTOL-MYERS SQUIBB COMPANY BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, OR ANY OTHER DAMAGES OF ANY NATURE WHATSOEVER, THAT MAY BE INCURRED BY THE USER OR ANY OTHER PARTY ARISING OUT OF OR IN CONNECTION WITH ANY USE OR PERFORMANCE OF THIS SOFTWARE OR ANY USE OF ANY DATA OR RESULTS GENERATED BY THE SOFTWARE, INCLUDING WITHOUT LIMITATION LOSS OF DATA AND LOST PROFITS OR REVENUES, WHETHER OR NOT DR. BRUCCOLERI AND BRISTOL-MYERS SQUIBB COMPANY HAVE BEEN ADVISED OF THE POSSIBILITY OF DAMAGES, AND USER HEREBY WAIVES, RELEASES, AND FOREVER DISCLAIMS ALL DAMAGES, CLAIMS, AND CAUSES OF ACTION IT MAY HAVE AGAINST DR. BRUCCOLERI AND BRISTOL-MYERS SQUIBB COMPANY WITH RESPECT TO ANY LOSSES, DAMAGES, COSTS, EXPENSES, AND LIABILITIES OF ANY NATURE THAT MAY BE INCURRED BY IT ARISING OUT OF OR IN CONNECTION WITH USER'S USE OF THE SOFTWARE AND ITS RELATED DOCUMENTATION. USER SHALL BE SOLELY RESPONSIBLE FOR ALL LOSSES, DAMAGES, COSTS, EXPENSES, AND LIABILITIES OF ANY NATURE INCURRED BY USER RESULTING FROM OR IN CONNECTION WITH USER'S USE OF THE SOFTWARE AND ITS RELATED DOCUMENTATION. THE USER UNDERSTANDS AND ACCEPTS THE ABOVE LIMITATIONS ON DAMAGES AND REMEDIES AS A CONDITION OF OBTAINING USE OF THE SOFTWARE AND RELATED DOCUMENTATION WITHOUT CHARGE.

NAME

rs6kfix – Fixes output from IBM RS6000 Fortran formatted I/O.

SYNOPSIS

rs6kfix [*input-file*]

DESCRIPTION

rs6kfix is a simple filter which changes output formats generated by the IBM RS6000 Fortran I/O library. Formatted output statements from this machine are unusual in that the leading zero in fractions whose magnitude is smaller than 1 is omitted. This filter puts that zero back. For example, **-.124** is converted to **-0.124**. This program is very handy when comparing results computed on an RS6000 with other machines.

An example usage would be

rs6kfix file1 | diff - file2 | ndiffpost cutoff=1.0e-5

SEE ALSO

ndiffpost(1)

ORIGIN

Robert E. Bruccoleri
Bristol-Myers Squibb Pharmaceutical Research Institute

NAME

avgtable – Average CONGEN tables.

SYNOPSIS**avgtable**

[-ave] [-sd] [*files ...*]

DESCRIPTION

avgtable averages the data in multiple CONGEN tables, as generated by the **WRITE unit** option, in the **BUILD TABLE** command, and writes the result into a new table.

An example usage would be

avgtable -ave table.out.1 table.out.2 table.out.3

Command line options

-ave Specify calculation of averages. This is the default.

-sd Specify calculation of standard deviations.

IMPLEMENTATION

The script is written in Perl.

ORIGIN

Robert E. Bruccoleri

Bristol-Myers Squibb Pharmaceutical Research Institute