[Public]

To: J3 J3/22-186
From: R. Bleikamp & JOR
Subject: Fortran 202y preprocessor functionality and syntax tutorial
Date: 2022-September-28
Reference: 22-181r1.pdf
More references (in no particular order): some references contain links to other documents
https://j3-fortran.org/forum/viewforum.php?f=23 , https://flang.llvm.org/docs/Preprocessing.html
https://github.com/j3-fortran/fortran_proposals/issues/65, https://gcc.gnu.org/onlinedocs/cpp/
https://gcc.gnu.org/onlinedocs/gfortran/Preprocessing-Options.html, http://www.daniellnagle.com/coco.html
and man pages for fpp/*cpp* for Cray, Intel, IBM, Nag, etc.

This tutorial is a status report of JoR's work on a preprocessor for F202y, some discussion points, and straw votes.

In this tutorial, the word *cpp* (in italics) refers generically to the C/C++ language preprocessor. The word preprocessor (absent *cpp* or other descriptive modifier) refers to the proposed Fortran preprocessor functionality forF202y.

The main sections below are the Summary, **What we expect to propose later**, survey of existing preprocessors, some comments on what we observed in the survey results, **discussion points, straw votes**, and additional notes. The topics in BOLD are what we hope will generate the most discussion.

Summary:

One of the proposed additions to F202y is a Fortran preprocessor similar to existing preprocessors provided by most vendors, based on *cpp*. JoRs goal is to provide a standard definition for a commonly available feature that is in widespread use today. Philosophically, we are trying to provide as much compatibility with existing Fortran preprocessor implementations as possible, but not going completely crazy. Introducing significant inconsistencies with all existing *cpp* based Fortran preprocessors would not usually be a good idea. We are not trying to provide all the functionality that *coco* or a new Fortran friendly design might be able to provide. Most of we expect to propose in the next year or so is existing practice in one or more well-known Fortran processors. Any attempts to provide a more *Fortran friendly* feature that no existing implementation provides would require substantial committee support to be considered. Comments from all interested parties are welcome. Send e-mail to the general mailing list, or to Rich (rich@bleikamp.net).

What we expect to formally propose in the next year or so

1. The syntax for preprocessor directives will be the same as for *cpp*.
2. Supported directives/features will include "__LINE__", __FILE__", #line, #ifdef, #ifndef, #define, #undef, #if, #include, ## (token concatenation without spaces), (and STRINGIFY())?. More may be added. No attempt to precisely define any of these directives/features has been started.
3. The Fortran preprocessor just works by default. It will be part of the Fortran language. We have not discussed whether a way to disable the preprocessor functionality must be provided by an implementation.
4. JoR believes the preprocessor should be implementable by a standalone process or within the compiler front end. This will improve the adoption rate of this feature in a timely manner.
5. Support for the preprocessor is required to be standard conforming with ISO/IEC 1539-1. This likely means the preprocessor must be defined in part 1 of the standard (a new clause?).
6. Although our syntax is based on a subset of *cpp*, we do not expect to exactly follow the current definition(s) of *cpp*, nor update the Fortran preprocessor in the future to maintain compatibility with the most recent definition of *cpp*.
7. Some additional *Fortran Friendly* functionality will be included. Mostly what existing implementations do.

Brief Survey of Fortran functionality provided by existing implementations (*-cpp or -fpp*, not *coco* or others)

This is only a partial list of "Fortran friendly" features. We have not started thinking about the finer points of cpp behavior.

Behavior	Vendor					
	Cray	Classic flang	gfortran	IBM	Intel	Nag
a) are ABC and Abc both replaced by #define ABC xxx?	N	N	N	N	N	N(5)
b) In fixed form, if the token "ABC" is defined to be "XYX", is "A B C" replaced?	N	N	N	N	N	N (6)
c) is "//" treated as a C++ comment and discarded?	N	N	N	N (9)	N	N
d) in fixed form, does token replacement cause source lines to be expanded beyond col. 72, causing significant characters to be ignored? (1)	N	Y	Y	Y	N	N
e) In fixed form, if the token C is replaced by Z, are Fortran comment lines turned into significant lines/statements, causing compilation errors?	Y	N	Y	Y	N (2)	N
f) In fixed form, if the token "A" is defined as "foo", will " A 10 " as an edit descriptor in a FORMAT statement be expanded causing an error?	Y	Y	Y	Y	N (2)	N(7)
g) Does "#define E x" cause compilation errors when "A = 1.0 E 4" is present in the source?	Y	Y	Y	Y	Y	Y(6)
h) are tokens in character literals replaced?	N	N	N	N	N	N
i) are tokens in Hollerith fields in FORMAT statements replaced?	N	Y	Y	N	N	N
j) are tokens replaced in OpenMP and other directives? (3)	Y	Y	Y	Y	Y	Y(7)
k) Are C comments in a #define discarded or ignored?	Y	Y	Y	Y	N	Y
l) are Fortran INCLUDE lines expanded / preprocessed	Y(4)	N	N	N	N	N(7)

(1) The Cray, Intel, and NAG compilers handle expanding a short token name into a longer string in a more Fortran friendly manner than other compilers. In fixed form, the conceptual expansion of significant text beyond column 72 did not cause that text to be treated as a comment. We did not determine how this was accomplished but observe that 2 obvious methods are (a) to introduce continuation lines as needed when significant text is expanded beyond column 72, or (b) to discard comments in columns 73+ before token replacement, do the token replacement, and enable the processor to use a longer line length (>72).

(2) Intel's compiler issues a warning about token replacement for the token "C" when the "C " is in columns 1-2 (fixed form) and token replacement does not occur. Other occurrences of "C" are replaced. Similarly for the token "A" in an edit descriptor such as " A 10" in a FORMAT statement (the edit descriptor is not subject to token replacement with some compilers). Flang has similar behavior for "C " in column 1, but no warning. Some other compilers token replace the "C ", possibly turning the comment into something else, and token replace an edit edit descriptor.

(3) In all compilers tested, a source file with "!$omp parallel" compiled with flags "-fopenmp -cpp -Dparallel=foo" caused "omp parallel" to become "omp foo". None of the compilers tested appeared to distinguish between OpenMP directive keywords and variable names in Openmp directives when performing token substitution.

(4) Cray's compiler does inline Fortran INCLUDE line files by default, even though the man page claims it does not. Ftn version 14.0.3. The man page claims an additional option must be enabled to preprocess INCLUDE line files.

(5) NAG provides a fpp flag to convert all uppercase non-literal characters to lower case. Presumably, this allows ABC and Abc to be treated as the same token for token replacement purposes.

(6) NAG documents an additional flag that treats insignificant blanks as truly insignificant for token expansion purposes. The default is that insignificant blanks are significant to fpp by default.

(7) according to NAG's documentation, the following items are excluded from token expansion:

- files included by Fortran INCLUDE lines;
- fpp and Fortran comments; (note, the initial "C" is column 1 **is** replaced by NAG's compiler, and OpenMP directives are subject to token replacement);
- IMPLICIT single letter specifications;
- FORMAT statements; But the compiler warned that " A " was not replaced in FORMAT (A 10)
  by (#define A foo)
- numeric and character constants.

(8) Most (probably all?) compilers will replace a Fortran keyword, in a keyword context. i.e.

```
#define do dodo
  do 10 i=1,10
   …
  end do
```

will cause compilation errors. Intel's compiler issues a warning when replacing a keyword.

(9) IBM provides a flag to treat "//" in a directive as a comment and discard it. Also to enable trigraph replacement, "!" comments in directives, and "&" as a preprocessor continuation marker.

Summary of the observed differences in the compilers tested above

So far, we have focused on Fortran friendly behavior, particularly with respect to token replacement. More testing is needed.

All the compilers tested are *cpp* like, with respect to token recognition and replacement, except as noted above.

The interesting differences are:

1) Some compilers do not do any token replacement (or at least avoid the obvious bad cases) in FORMAT statements.
2) Some of the compilers handle token replacement that pushes significant text beyond column 72 better than others.
3) Some compilers treat a "C " or "c " in columns 1-2 as special, and don't do token replacement on the "C "/"c ". Others replace the "C", and that line is no longer a comment.
4) One compiler allows a Fortran INCLUDE line to be treated as a #include preprocessor directive.
5) Most compilers discard a C /* xxx */ comment on the end of a #define line, so the Fortran processor does not see it. Others pass it thru, causing compilation errors.

Features that are the same in all tested compilers (in the default mode):

1) Token replacement is usually very *cpp* like, with little support for fixed form Fortran oddities.
2) Case of letters in a token is always significant, unlike Fortran.
3) "//" in a preprocessor directive is not discarded (not treated as a C++ comment).
4) Tokens in an OpenMP directive are subject to token replacement.

Some observations about existing practice:

The first 5 items below, while not all *Fortran friendly*, reflect existing default practice in every implementation tested.

1) The case of the characters in a token are significant when determining what text in a program will be replaced. i.e. "#define ABC x" will cause "ABC" to be replaced, but not "Abc".

2) In fixed form, insignificant blanks are significant in token replacement. "#define ABC xxx" will cause "ABC" to be replaced but not "A B C".

3) Character literals (text within "s or 's) are NOT subject to token replacement within the quoted text.

4) Fortran comment lines that begin with !$ or C$ are subject to token replacement.

5) In fixed form, tokens such as "E" or "D", when delimited by insignificant blanks, are subject to token replacement, even in a real literal constant (i.e. "A = 1.0 D 4"). The rules for delimiting preprocessor tokens are usually (always?) *cpp* like, not Fortran like.

A final comment on existing practice:

1) In fixed form, expansion of tokens probably should not cause significant text to be conceptually pushed beyond column 72 and then treated as a comment. Only Cray, Intel, and NAG behave in a *Fortran friendly* manner today. flang and gfortran will sometimes push significant (non-comment) text beyond column 72, usually resulting in fatal errors. Usually, we wouldn't be overly concerned about

Discussion points:

1) In the Summary above, we said

   "The goal of this feature is to provide a standard definition for a commonly available feature that is in widespread use today." and

   "Most of we propose below is existing practice in one or more well-known Fortran processors. Any attempts to provide a more *Fortran friendly* feature that no existing implementation provides would require substantial committee support to be considered.".

   Based on these guiding principles (feel free to argue about the guiding principles), we invite discussion/comments and a straw vote on (2) below. This is the most useful information the committee can provide now, to avoid future wasted effort. Detailed lists of what should or shouldn't be handled by the preprocessor are welcome, but not useful yet.

2) In general, when deciding how *Fortran friendly* to be, four obvious options are:

   a. Just use an existing *cpp*, as is, with **no** changes to be more *Fortran friendly*. No compiler we tested provides a preprocessor this *Fortran un-friendly*. JoR does **not** recommend option (a).

   b. Start down the slippery slope, providing as many *Fortran friendly* features as are provided today by one of more vendors. This might include:

      i. Intelligent handling of token replacement which pushes significant text beyond column 72 in fixed form. The text conceptually pushed beyond column 72 is still significant, not a comment.

      ii. Skipping token replacement in a several contexts, including character literals, FORMAT statements (i.e. Hollerith fields and edit descriptors), and maybe not token replacing a "C" in column 1.

      iii. Do token replacement in OpenMP/ACC/other "directives". Essentially do token replacement in Fortran comments, except for the "C" in column 1 in fixed form,.

      But not include doing case insignificant token replacement, (ABC and AbC are different tokens), nor squeezing out insignificant blanks before identifying what is a token.

      i.e. in "1.0 D 0", the "D" would be a token, but not in "1.0D0".

   c. Slide a little further down the slope, adding some more *Fortran friendly* support, possibly including:

      i. Treat tokens as case insensitive, and squeeze out insignificant blanks in fixed form, so that "ABC" would be the same token as "A b c" in fixed form, and "ABC" and "AbC" are the same token in free form. This requires a minimal lexical scanner in a stand-alone preprocessor, but not a full blown traditional ad-hoc lexical scanner.

      ii. Don't token replace the "type" letters in an IMPLICIT statement.

      iii. Don't token expand token starting in column 6 in fixed form. (this could be part of (b) above)

      iv. Your favorite pet peeve?

   d. Do most everything we can think of. In a stand alone preprocessor, this essentially requires a full blown ad-hoc Fortran lexical scanner (and possibly a parser?) to get everything correct. If implemented within

an existing front end, several existing compilers would encounter difficulties (those that use a backup parser). The supported features might include:

i. Avoiding replacement of Fortran keywords when used as a keyword (but do replace a keyword when it is a variable name).

ii. Supporting Fortran expression syntax and constant expressions, instead of C like *cpp* expression syntax. Datatypes/operators/intrinsics supported TBD. This would have to be user selectable mode, to avoid upsetting all existing Fortran preprocessor users. It also adds a lot of implementation cost for a stand-alone preprocessor.

This option will likely incur noticeable overhead compared to the other options.

JoR does **not** recommend option (d).

Straw Votes: you may vote for as many options as you wish:

SV1: do you believe option (a) above is acceptable?

SV2: do you believe option (b) above is acceptable?

SV3: do you believe option (c above is acceptable?

SV4: do you believe option (d) above is acceptable?

Informational Note

Gary Klimowicz is preparing a detailed list of cpp features and Fortran-friendly behaviors that a preprocessor could provide. I have arbitrarily grouped some of these into categories below, but JoR hasn't discussed them yet. They are provided merely as guidance as to what **<u>might</u>** be included in the various options listed in the straw vote above

Group (a): very cpp like preprocessor

- The usual cpp directives, and cpp-ike token replacement.

Feature	Flavor	In?	Source
# non-directive	directive		cpp
# if	directive		cpp
# ifdef	directive		cpp
# ifndef	directive		cpp
# elif	directive		cpp
# else	directive		cpp
# endif	directive		cpp
# include	directive		cpp
# define id replacement-list	directive		cpp
# define id (id-list) replacement-list	directive		cpp
# define id (...) replacement-list	directive		cpp
# define id (id-list , ...) replacement-list	directive		cpp
# undef	directive		cpp

# line	directive		cpp
# error	directive		cpp
# pragma	directive		cpp
# new-line	directive		cpp
#	operator		cpp
##	operator		cpp
defined	operator		cpp
!	operator		cpp
?	expr		
__DATE__	macro		cpp
__FILE__	macro		cpp
__LINE__	macro		cpp
__STDFORTRAN__	macro		cpp-ish
__STDFORTRAN_HOSTED__	macro	N	cpp-ish
__STDFORTRAN_VERSION__	macro		cpp-ish
__TIME__	macro		
STRINGIFY	macro	N	Clu1
__SCOPE__	macro	N	Clu1, Lio1
__VENDOR__	macro	N	Clu1

Group (b): all of group (a) and essentially all Fortran friendly behavior that at least one existing Fortran processor provides today, sometimes with an additional compiler flag or two.

FF: Right margin clipping at column 72 or 132	Fortran		Fla1
FF: No right margin clipping on directive lines	Fortran		Fla1
FF: Spaces significant in determining tokens	Fortran		Fla1
FF: No expansion of C (or D) in column 1	Fortran		Ble1
FF: Expansion of C (or D) in column 1	Fortran		Ble1, Fla1
FF: Expanded text reflects fixed-format rules	Fortran		Fla1
FF: Strip Column 1 C comments from expanded text			Fla1
No expansion in strings	Fortran		Ble1, Fla1
No expansion in Hollerith			Ble1
Expansion in comments			Ble1
Strip C-style /* ... */ comments			Fla1
Expand INCLUDE lines as if #include	Fortran		Fla1, Jor1

Group (c): all of group (b) plus:

FF: No expansion of column 6	Fortran		Kli1

C-style line continuations			Fla1
No expansion in IMPLICIT single-character specifiers			Ble1
No expansion of FORMAT specifiers			Ble1, Fla1
Expansion in directives (e.g., OpenMP)			Ble1
Comment lines in definitions with continuation lines			Fla1

** Notes

- FF :: Regarding fixed-form input and corresponding output.

Additional Notes

The following functionality that might be included in a preprocessor, as described in the straw vote above, might deserve some additional discussion.

1) Should all Fortran comment lines that start with a “C ” or “c ” in columns 1-2 (fixed form) be subject to token replacement of that initial “C” or “c” in column 1, and then potentially not be a comment? If not a comment, how does it integrate with surrounding (possibly continued) statements? To simplify implementations, we are leaning towards not allowing a “C “ or “c “ in columns 1-2 (while in fixed form) to be subject to token replacement. The rules for how such a line would integrate into surrounding statements / continuation lines might cause implementation headaches for some vendors.
2) What should we do about C comments (/* xxx */) on a #define statement? Discard them (ignore them?) or pass them on to the Fortran front end (causes errors typically)? Most existing implementations discard them, and I don’t see any practical use for retaining them in the pre-processed text. If we did retain them, should the compiler treat them as a comment?
3) Should filenames specified in an INCLUDE line (the Fortran INCLUDE, not #include) be conceptually inlined and subject to preprocessor actions like Cray? Or possibly optionally?
4) Should FORMAT statements be subject to token replacement? Intel and NAG seem to NOT allow token replacement in FORMAT statements. We are leaning towards NOT allowing token replacement in FORMAT statements. Is there any reason to do token replacement in FORMAT statements.
5) What other directive based stuff exists that does not use “!$”, “C$”, or “*$” *xxx* as a sentinel to identify a directive. OpenMP and ACC use !$omp and !$acc (similarly for C$ and *$ in fixed form?
6) Should comments of all types be subject to the usual token expansion rules, except for the “C “ in columns 1-2? This would allow any directive based stuff to work like the !$ and c$ sentinels do for openmp and ACC in all implementations today.
7) What have we forgotten/ignored? *Fortran specific* behaviors that might be contentious are of the most interest.