Fortran preprocessor requirements

INCITS/Fortran JoR

[2023-10-24 Tue 11:27]

1 Summary

We summarize the potential requirements for the Fortran 202y preprocessor.

Each heading is of the form `[shorthand-name] One-line description`. There may be an additional tag (category) on the line (such as `:output:`). The section body contains information on

- Where the requirement came from in Fortran discussions and posts
- Current status (TBD, accepted by JoR, accepted by WG5, rejected by JoR, rejected by WG5, etc.)
- Where the reference appears in the C 2018 standard *ISO/IEC 9899:2018 Information technology – Programming languages – C*. (We will update for 2024 when ratified.) We write these references with
  - an opening square bracket `[` and the letter `C`, followed by
  - the section marker `§` and the section number, followed by
  - (optionally) a `¶` followed by the paragraph number, followed by
  - (optionally) a bullet symbol · and the bullet number or letter, followed by
  - (optionally) the letter `f` and the footnote number, followed by
  - (optionally) the section title for redundant context, followed by
  - a closing square bracket `]`.

  For example, `[C§5.1.1.2¶1·2 Translation phases]` references section 5.1.1.2 of the C 2018 standard, paragraph 1, second bullet point. This is in the "Translation phases" section.

- Where the reference appears in the Fortran 2023 standard *ISO/IEC FDIS 1539-1 Information technology — Programming languages — Fortran*. We write these references with
  - an opening square bracket `[` and the string `F2023`, followed by
  - the section marker `§` and the section number, followed by one of
    - an `R` followed by a rule number, or
    - a `C` followed by a constraint number, or
    - a ¶ followed by the paragraph number, followed optionally by a bullet symbol · and the bullet number or letter
  - (optionally) the section title for redundant context, followed by
  - a closing square bracket `]`.

  For example, `[F2019§13.2.1R1301 FORMAT statement]` references section 13.2.1 of the Fortran 2023 standard, rule 1301. This is in the "FORMAT statement" section.

1.1 Translation phases

Similar to the C standard, we define phases of text processing related to preprocessing. The preprocessor performs the following phases on the input source program before the processor transforms the program for use.

This requires implementations to behave as if these separate preprocessing phases occur, even though they are typically folded together in practice. Source input need not necessarily be stored as files, nor need there be any one-to-one correspondence between these entities and any external representation. The description is conceptual only, and does not specify any particular implementation.

1. Continuation lines in the input source are interpreted, producing a sequence of logical lines (introducing new-line characters for end-of-line indicators). [C§5.1.1.2¶1·1-2]

2. The preprocessor decomposes the logical lines into preprocessing tokens and sequences of white-space characters and comments. This input shall not end in a partial preprocessing token. New-line characters are retained. For the proper handling of compiler directives, comments are retained. Whether each nonempty sequence of white-space characters other than new-line is retained or replaced by one space character is implementation-defined. [C§5.1.1.2¶1·3]

3. Preprocessing directives are executed and macro invocations are expanded. A #include preprocessing directive causes the named header or source file to be processed from phase 1 through phase 3, recursively. All preprocessing directives are then deleted. [C§5.1.1.2¶1·4]

4. Each preprocessing token is converted into a token for subsequent handling by the processor.

2 Phase 1 Continuation handling

2.1 [c-backslash] C-style \ line continuations in directives

- **Status:** TBD
- **Source:** Fla1
- **References:** [C§5.1.1.2¶1·2 Translation phases]

In fixed-form and free-from source code, delete a backslash \ immediately followed by a newline character.

From The C standard:

> Each instance of a backslash character (\) immediately followed by a new-line character is deleted, splicing physical source lines to form logical source lines. Only the last backslash on any physical source line shall be eligible for being part of such a splice. A source file that is not empty shall end in a new-line character, which shall not be immediately preceded by a backslash character before any such splicing takes place.

2.2 [fortran-cont-fixed] Process Fortran line continuation in directives

- **Status:** TBD
- **Source:** Che1
- **References:**

In fixed-form input, a character in column 6 that is not blank or the digit zero continues the line with the previous line, even if the previous line is a directive line, or the continuation of a directive line.

2.3 [fortran-cont-free] Fortran line continuation in directives in free form

- **Status:** TBD
- **Source:** Che1
- **References:** [C§6.10 Preprocessing directives]

In free-form input, an `&` character as the last character on a directive line indicates the directive continues on the next line. The handling of the continuation is as described in [F2023§6.3.2.4].

2.4 [fortran-cont-free-trim] Fortran line continuation in directives in free form

- **Status:** TBD
- **Source:** Che1
- **References:** [C§6.10 Preprocessing directives]

In free-form input, an `&` character as the last character on a directive line indicates the directive continues on the next line. When the first non-blank character on the next line is also an `&`, the characters between the ampersands are deleted.

2.5 [c-comment-strip] Strip C-style `/* ... */` comments

- **Status:** TBD
- **Source:** Fla1
- **References:** [C§6.10¶5 Constraints], [C§6.10.1¶6f172 Semantics]

2.6 [comment-definition-cont] Comment lines in definitions with continuation lines

- **Status:** TBD
- **Source:** Fla1
- **References:** [C§6.10 Preprocessing directives]

2.7 [comment-bang] Recognize the comment character '!' in directives

- **Status:** TBD
- **Source:** Che1
- **References:** [C§6.10 Preprocessing directives]

3 Phase 2 Tokenization

3.1 [tokens-case-insensitive] Case insensitive tokens

- **Status:** TBD
- **Source:** Fla1

Fortran is not case-sensitive. The preprocessor is not case-sensitive when recognizing identifiers. The text fragment

```
#define abc XYZ
#define ABC foo
      subroutine abc
```

should expand to

```
      subroutine foo
```

If the preprocessor were case sensitive, we would have behavior like

```
#define ABC var_1
#define abc var_2
      abc = ABC + 1     ! Normally, Fortran treats these as the same identifier
```

expanding to

```
      var_2 = var_1 + 1    ! These identifiers are now different
```

We should expect it to expand to

```
      var_2 = var_2 + 1    ! Only the second definition matters
```

3.2 [identifiers-span-continuations] Identifier tokens are not broken by line continuations

- **Status:** TBD
- **Source:** Kli1, Jor4
- **References:** [F2023§6.2.2C601]

  In fixed-form, there are only 66 characters available for statement text (columns 7-72). The maximum length of an identifier is 63 characters. It is not practical to have identifiers end at a fixed-form line boundary at column 72.

3.3 [spaces-end-token] Spaces significant in determining tokens

- **Status:** TBD
- **Source:** Fla1
- **References:**

In order to simplify the preprocessor tokenization, spaces are significant, even in fixed-form source.

4 Phase 3 Directive processing

4.1 [no-fixed-form-column-6] A # in column 6 in fixed-form is not a directive

- **Status:** TBD
- **Source:** Kli1, Jor4
- **References:**

This is seen in some existing projects that use `#` in column 6 for a conventional continuation line. These are the Fortran files in fortran-examples that have `#` in column 6. Note that some expect to run through the preprocessor (extension is `.F`).

ALBUS_ionosphere@twillis449/FORTRAN/IRI/igrf.f
CMAQ@USEPA/POST/sitecmp_dailyo3/src/process.F
E3SM@E3SM-Project/components/mpas-ocean/src/mode_forward/mpas_ocn_time_integration_si.F
Genetic-Algorithm-for-Causeway-Modification@stevenmeyersusf/Code/genmain.f
MCFM-RE@lcarpino/src/Parton/eks98r.f
MITgcm@MITgcm/pkg/openad/externalDummies.F
NCEP_Shared@GEOS-ESM/NCEP_w3/w3ersunb.f
OEDGE@ORNL-Fusion/lim3/comsrc/sysaix.f
PublicRelease_2020@FLOSIC/flosic/scan.f
STELLOPT@PrincetonUniversity/LIBSTELL/Sources/NCLASS/nclass_mod.f
ShirleyForQE@subhayanrc/yambo-stable/src/real_time_common/RT_driver.F
cernlib@apc-llc/2005/src/graflib/higz/imac/f_readwi.F
cernlib@apc-llc/2006/src/graflib/higz/imac/f_readwi.F
cfdtools@nasa/app/traj_opt/numerics.f
cfdtools@nasa/lib/searchlib/hsortcc.f
dynamite@dynamics-of-stellar-systems/legacy_fortran/galahad-2.3/src/ma27/ma27d.f
forestclaw@ForestClaw/applications/clawpack/euler/2d/rp/rpn2euq3.f
hompack90@vtopt/src/MAINP.f
legacy-mars-global-climate-model@nasa/code/cmp3out.f
nosofs-NCO@ioos/sorc/SELFE.fd/utility/Combining_Scripts/combine_outHA.f
nwchem@nwchemgit/src/nwpw/band/lib/psi/cpsi_KS.F
nwchem@nwchemgit/src/tce/mrcc/tce_mrcc_energy.F
pyOpt@madebr/pyOpt/pySLSQP/source/slsqp.f
pyclaw@clawpack/development/rp_approaches/rpn2_euler_5wave.f
scream@E3SM-Project/components/mpas-ocean/src/mode_forward/mpas_ocn_time_integration_si
starlink@Starlink/applications/echomop/ech_kdhsubs.f
starlink@Starlink/applications/obsolete/iras90/misc/ffield.f
starlink@Starlink/thirdparty/caltech/pgplot/examples/pgdemo17.f

4.2 [non-directive] # non-directive

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10 Preprocessing directives]

4.3 Conditional inclusion

4.3.1 [#if] # if *constant-expression*

- **Status:** TBD

- **Source:** cpp
- **References:** [C§6.10.1 Conditional inclusion]

4.3.2 [#ifdef] # ifdef *identifier*

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.1 Conditional inclusion]

4.3.3 [#ifndef] # ifndef *identifier*

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.1 Conditional inclusion]

4.3.4 [#elif] # elif *constant-expression*

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.1 Conditional inclusion]

4.3.5 [#else] # else

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.1 Conditional inclusion]

4.3.6 [#endif] # endif

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.1 Conditional inclusion]

4.4 Source file inclusion

4.4.1 [#include] # include *char-literal-constant*

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.2 Source file inclusion], [F2023§7.4.4.3 Character literal constant], [F2023§6.4 Including source text]

4.4.2 [#include-computed] # include *pp-tokens*

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.2 Source file inclusion]

4.5 Macro replacement

4.5.1 [#define-id] # define id replacement-list

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.3 Macro replacement]

4.5.2 [#define-id-function] # define id (id-list) replacement-list

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.3 Macro replacement]

4.5.3 [#define-id-0-varargs] # define id (...) replacement-list

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.3 Macro replacement]

4.5.4 [#define-id-n-varargs] # define id (id-list , . . .) replacement-list

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.3 Macro replacement]

4.5.5 [#undef] # undef

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.3.5 Macro replacement]

4.6 Line control

4.6.1 [#line] # line

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.4 Line control]

4.7 Error directive

4.7.1 [#error] # error

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.5 Error directive]

4.8 Pragma directive

4.8.1 [#pragma] # pragma

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.6 Pragma directive]

4.9 Null directive

4.9.1 [#null] # newline

- **Status:** TBD
- **Source:**
- **References:** [C§6.10.7 Null directive]

4.10 Additional requests

4.10.1 [#show] # show

- **Status:** TBD
- **Source:** Lio3
- **References:** [C§6.10 Preprocessing directives]

  #show prints a table of all the macros; and a "#show namea nameb name*" does the same with just the listed names, but allows simple globbing. Simple and sometimes useful for distinguishing between output files when actually retaining the intermediate files. All the output is written to the output file with lines starting with ! (not standard, but even a lot of pre-f90 compilers allowed ! as a comment)

  so the output is still valid Fortran.

4.10.2 [#import] # import VARNAME

- **Status:** TBD
- **Source:** Lio3
- **References:** [C§6.10 Preprocessing directives]

  which imports an environment variable as if it had been defined with -DVARNAME=VALUE

4.10.3 [#output] # output filename [–append]

- **Status:** TBD
- **Source:** Lio3
- **References:** [C§6.10 Preprocessing directives]

  which makes it easy to have a single file that outputs Fortran, C, markdown ... sections but would complicate a preprocessor being "inline" in the compiler, which I hope is an expected feature of a standard preprocessor, thus being able to eliminate having to generate (or at least retain) intermediate files, being able to define reusable blocks of plain test and reuse them or loop over them applying an optional filter that can convert them all to comments, convert them all the a character variable definition, or convert them to WRITE statements. Makes maintaining comments and help text a lot easier, as you can just type it as plain text, for example. I would be content with just cpp-like functionality, but those are features I use a lot cpp(1) does not do, except typically (not in all fpp flavors) block comments are supported. I think a #import would be useful and simple though. Perhaps a group concensus would be it is problematic, making sure it is not inadvertently the wrong value, ... so not sure even that would make it into a first-generation standard utility.

5 Expressions

5.1 [#-operator]

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.3.2 The # operator]

5.2 [##-operator]

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.3.3 The ## operator]

5.3 [defined-operator] `defined` *identifier*

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.1 Conditional inclusion]

5.4 [defined-operator] `defined (` *identifier* `)`

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.1 Conditional inclusion]

5.5 [bang-operator] `!`

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10 Preprocessing directives]

5.6 [c-expressions] C-style expressions

- Source :::
- **Status:** TBD
- **References:** [C§6.10 Preprocessing directives]

This is problematic, in that there are many files in the wild that have the `!` not operator in `#if` and `#elif` directives. This is a Fortran comment character.

5.7 [fortran-expressions] Fortran-style expressions

- Source :::
- **Status:** TBD
- **References:** [C§6.10 Preprocessing directives]

6 Expansion

6.1 [fixed-no-expand-c-col-1] No expansion of `C` in column 1

- **Status:** TBD
- **Source:** Ble1
- **References:** [C§6.10 Preprocessing directives]

In fixed-form, `C` in column 1 indicates a comment. No identifier that begins with a `C` in column is expanded.

6.2 [fixed-no-expand-d-col-1] No expansion of `D` in column 1

- **Status:** TBD
- **Source:** Ble1, Fla1
- **References:** [C§6.10 Preprocessing directives]

In fixed-form, `D` in column 1 is a common extension to indicate a comment. No identifier that begins with a `D` in column is expanded.

6.3 [fixed-no-expand-col-6] No expansion of column 6

- **Status:** TBD
- **Source:** Kli1
- **References:** [C§6.10 Preprocessing directives]

In fixed-form, a character in column 6 that is not blank or zero indicates a continuation line. No identifier that begins in column 6 is expanded.

[Note that since we define expansion to occur after continuation handling, this requirement is not necessary.]

6.4 [fixed-strip-col-1-comments] Strip column 1 `C` comments from expanded text

- **Status:** TBD
- **Source:** Fla1
- **References:** [C§6.10 Preprocessing directives]

6.5 [pass-comments] Pass comments from expanded text to the processor

- **Status:** TBD
- **Source:** Fla1
- **References:**

This is necessary to pass comments with directives to the Fortran processor.

6.6 [no-expand-string] No expansion in strings

- **Status:** TBD
- **Source:** Ble1, Fla1
- **References:** [C§6.10 Preprocessing directives]

String constants are output without being examined for macro expansion.

6.7 [no-expand-hollerith] No expansion in Hollerith

- **Status:** TBD
- **Source:** Ble1
- **References:** [C§6.10 Preprocessing directives]

No expansion occurs in the string contained in a Hollerith constant.

6.8 [no-expand-implicit-char-list] No expansion in `IMPLICIT` single-character specifiers

- **Status:** TBD
- **Source:** Ble1
- **References:**

The letters in an `IMPLICIT` statement are not considered for macro expansion.

Note that this implies the preprocessor recognizes `IMPLICIT` statements.

6.9 [no-expand-format] No expansion in `FORMAT` specifiers

- **Status:** TBD
- **Source:** Ble1, Fla1
- **References:** [F20123§13.2.1R1301]

In `FORMAT` statements, there is no macro expansion in the *format-specification.*. Note that this implies the preprocessor recognizes `FORMAT` statements.

6.10 [expand-comments] Expansion in comments

- **Status:** TBD
- **Source:** Ble1
- **References:**

Outside the exceptions noted elsewhere, the preprocessor expands macros in the text of comments.

[This may be the way the preprocessor preserves directives (such as OpenMP and OpenACC) in the program. We have heard from at least one J3 member that this is on important feature. The good news is that expanding macros always will handle directives, without having to do special processing for all manner of directives. The other good news in that comments without directives are ignored by the processor, so we don't care (maybe) how they are mangled.]

6.11 [expand-directives] Expansion in directives (e.g., OpenMP)

- **Status:** TBD
- **Source:** Ble1
- **References:**

This is problematic, as how does the preprocessor know which comments are directives?

6.12 [preprocess-fortran-include] Expand `INCLUDE` lines as if `#include`

- **Status:** TBD
- **Source:** Fla1, Jor1, JoR4

Assuming the preprocessor is a mandatory part of the Fortran standard, preprocessor directives are allowed in the file specified in a Fortran `INCLUDE` line. Therefore, the preprocessor should process the `INCLUDE`-ed file as if it had been invoked via the `#include` directive.

Otherwise, where will the handling of directives the included file be handled, and how can it use any of the macro definitions available at the time the `INCLUDE` statement is encountered. (It is likely to be included in multiple places in the application.)

6.13 [expand-kind-param] Expand macro names in *kind-param* in literal constants

- **Status:** TBD
- **Source:** JoR3
- **References:** [F2023§7.4.3.1R709 Integer type], [F2023§7.4.3.2R714 Real type], [F2023§7.4.4.3R724 Character literal constant], [F2023§7.4.5R725 Logical type]

If the *kind-param* is a *scalar-int-constant-name* following the underscore in an *int-literal-constant*, *real-literal-constant*, and *logical-literal-constant*, that constant name is subject to macro expansion. This needs to be explicit, as otherwise the preprocessor might treat `_kind-name` as an identifier, as many preprocesser predefined macro names begin with an underscore.

In a *char-literal-constant*, if the *kind-param* preceding the underscore (`_`) is a *scalar-int-constant-name*, that constant name is subject to macro expansion. This needs to be explicit, as otherwise the preprocessor might treat `kind-name_` as an identifier.

7 Output form

7.1 [fixed-clip-input] Right margin clipping at column 72

- **Status:** TBD

- **Source:** Fla1
- **References:**

7.2 [fixed-no-directive-clip] No right margin clipping on directive lines

- **Status:** TBD
- **Source:** Fla1
- **References:**

7.3 [fixed-output-conform] Expanded text reflects fixed-format rules for fixed-form input

- **Status:** TBD
- **Source:** Fla1, Jor4
- **References:**

If the processor produces output corresponding to the preprocessed input (e.g., via something like a `-E` option), it must produce valid fixed-form source code. This may require re-flowing the preprocessed output to the 72-column boundary.

8 Predefined macros

8.1 [file-process-date] `__DATE__`

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.8 Predefined macro names]

8.2 [file-name-context] `__FILE__`

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.8 Predefined macro names]

8.3 [line-number-context] `__LINE__`

- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.8 Predefined macro names]

8.4 [fortran-conform] `__STDFORTRAN__`

- **Status:** TBD
- **Source:** cpp-ish
- **References:** [C§6.10.8 Predefined macro names]

8.5 [hosted-implementation] `__STDFORTRAN_HOSTED__`

- **Status:** Not accepted
- **Source:** cpp-ish
- **References:** [C§6.10.8 Predefined macro names]

8.6 [fortran-version] `__STDFORTRAN_VERSION__`

- **Status:** TBD
- **Source:** cpp-ish
- **References:** [C§6.10.8 Predefined macro names]

8.7 [file-process-time] `__TIME__`

- Source :::
- **Status:** TBD
- **Source:** cpp
- **References:** [C§6.10.8 Predefined macro names]

8.8 [stringify-macro] `STRINGIFY`

- **Status:** Not accepted
- **Source:** Clu1
- **References:**

8.9 [scope-macro] `__SCOPE__`

- **Status:** Not accepted
- **Source:** Clu1, Lio1
- **References:**

8.10 [vendor-macro] `__VENDOR__`

- **Status:** Not accepted
- **Source:** Clu1
- **References:**

8.11 [no-undecorated-std-definitions] undecorated names (no _) defined by preprocessor

- **Status:** TBD
- **Source:** Lio2
- **References:**

9 Sources

- cpp: *cpp* if in the C standard (2018), *cpp-ish* if in C standard, but "Fortranized".
- Ble1: JoR Email threads from Rich Bleikamp re: tutorial *[2022-08-08 Mon 21:34]*.
- Che1: Email from Daniel Chen to JoR *[2022-07-29 Fri 11:08]*.
- Clu1: Email from Tom Clune *[2022-08-01 Mon 10:48]*.

- Fla1: LLVM Flang Preprocessing.md [https://github.com/llvm/llvm-project/blob/main/flang/docs/Preprocessing.md]
- Jor1: JoR meeting on preprocessors *[2022-08-22 Mon 10:00]*.
- Jor2: JoR meeting on preprocessors *[2022-09-20 Tue 13:00]*.
- Jor3: JoR meeting on preprocessors *[2023-11-07 Tue 12:00]*.
- Jor4: JoR meeting on preprocessors *[2022-12-06 Tue 12:00]*.
- Kli1: Private communication in his head.
- Lio1: Email from Steve Lionel *[2022-08-01 Mon 13:52]*.
- Lio2: JoR discussion forum https://j3-fortran.org/forum/viewtopic.php?p=561
- Lio3: JoR discussion forum https://j3-fortran.org/forum/viewtopic.php?p=562

10 References

- Jor email re: cpp tutorial for October meeting?
- INCITS+ISO+IEC+9899+2018+(2019)
- INCITS+ISO+IEC+JTC 1+1539-1+(2023)
- LLVM Flang Preprocessing.md