# FPP Phase 0 — Conjoin Lines

INCITS/Fortran JoR

November 23, 2024

## 1 Introduction

The INCITS/Fortran committee has discussed several times in the Plenary meetings, and the JoR subgroup has met several times, to lay down formal requirements for the proposed Fortran preprocessor (FPP) for Fortran 202y (presumably 2028).

This paper lists the requirements and syntax for Phase 0 of the preprocessor, the Line Conjoiner.

### 1.1 Translation phases

The C standard [ISO24] defines eight translation phases. These phases each perform a well-defined set of operations on the C source code and intermediate representations. They define a processing pipeline where one phase transforms its input in some way, and its output becomes the input to the next phase.

While these phase descriptions explain how C compilers *should behave*, they do not prescribe how C compilers *should be written*.

We do the same for Fortran. For FPP, though, we are only concerned with phases through interpreting preprocessor directives.

#### 1.1.1 Phase 1: Remove continuations

For fixed-form Fortran source, follow the column-6 conventions to produce a sequence of logical lines.

For free-form Fortran source, follow the `&` conventions to produce a sequence of logical lines.

In either form, remove continuations from directive lines (those lines beginning with `#`).

### 1.1.2 Phase 2: Process comments

For fixed and free-form source, translate comment-based directives (such as `!dir$`, `!omp$`, `!acc$`, and `CDIR$`, `COMP$`, and `Cacc$`) into some kind of formal pragma (such as a `#pragma` directive). Replace other comments with spaces.

### 1.1.3 Phase 3: Tokenize the source into preprocessing tokens

The output from Phase 3 is converted to preprocessor tokens according to the rules defined in "On Fortran source form" above.

### 1.1.4 Phase 4: Execute preprocessor directives

Preprocessing directives in the output from Phase 4 are executed. As in C, the execution of preprocessor directives and interpretation of macro definition and expansion is a *token-replacement* process, not a *text replacement* process.

Macros are expanded in Fortran source.

Source code is included, excluded, or modified based on the directives.

## 2 High-level requirements

The Line Conjoiner phase eliminates all continuation lines in the input source files. It takes as input source code in physical lines and produces *logical lines* of source code as follows:

- For preprocessor directive lines (beginning with `#`) it removes the `\ new-line` line continuations.
- For fixed-form Fortran source, it removes continuations introduced with characters in column 6.
- For free-form Fortran source, it removes continuations introduced by `&` at the end of line.

# 3 Detailed requirements

Each requirement is a heading of the form "One-line description". For each requirement, we include

**ID** A requirement unique identifier in square brackets `[]`.

**Status** The current status (TBD, JoR yes, JoR no, WG 5 yes, WG 5 no, etc.).

**Sources** Where the requirement came from in normative references (such as the C standard), J3 Fortran discussions and posts.

## 3.1 Requirement sources

These requirements came from the following sources.

**[C2024]** The C standard [ISO18].

**[F2023]** The Fortran standard [ISO23].

**[G14]** The C Preprocessor [SW24].

**facpp** The C standard, but made Fortran-aware.

**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]*.

**gak** Gary Klimowicz as he wrote these specifications.

**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]*.

**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.

## 3.2 Directives

### 3.2.1 Directive lines have # as the first non-blank character on the line

| ID | Status | Sources |
|---|---|---|
| [dir-firstnonblank-char-pound] | TBD | [C2023§5.1.1.2¶1·2 Translation phases]<br>[G14§1.4¶4 The preprocessing language] |

Rationale: [C§6.10¶2 Description].

### 3.2.2 Whitespace is allowed before the #

| ID | Status | Sources |
|---|---|---|
| [dir-whitespace-before-pound] | TBD | [C2024§6.10¶2 Description],<br>[G14§1.4¶4 The preprocessing language] |

### 3.2.3 Whitespace is allowed after the #

| ID | Status | Sources |
|---|---|---|
| [dir-whitespace-after-pound] | TBD | [G14§1.4¶4 The preprocessing language] |

### 3.2.4 4095 characters allowed in a directive logical line

| ID | Status | Sources |
|---|---|---|
| [logical-line-length-4095] | TBD | [C2024§5.2.4.1¶1•15 4095 characters in a logical source line],<br>[G14§11.2¶2•7 Number of characters on a logical source line] |

### 3.2.5 Directive lines are not bound by fixed-form conventions

Rationale: The preprocessor directives are a sub-language within the Fortran standard. In accordance with CPP, these lines are neither fixed-form nor free-form Fortran.

#### 3.2.5.1 in a fixed-form Fortran source file, Column 6 in a directive line does not mark a directive continuation

| ID | Status | Sources |
|---|---|---|
| [dir-column-6-not-cont] | TBD | [C§5.1.1.2¶1·2 Translation phases], [G14§] |

Rationale: preprocessor lines are not Fortran source lines.

### 3.2.6 Recognize C-style `\new-line` line continuations in directives

| ID | Status | Sources |
|---|---|---|
| [c-backslash-dir] | TBD | [C§5.1.1.2¶1·2 Translation phases], [G14§1·3 Fixed form right margin clipping] |

In a directive line fixed-form and free-from source code, delete a backslash \ immediately followed by a new-line character. The following line continues the directive line.

Rationale: 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.

Such lines already appear in our corpus of Fortran source programs.

### 3.2.7 No comment lines in definitions with continuation lines

| ID | Status | Sources |
|---|---|---|
| [comment-definition-cont] | TBD | [G14§], [C§6.10 Preprocessing directives] |

Fortran source lines with continuations may have comment liens between the continuation lines.

Rationale: CPP allows comments an the same line as preprocessor directives. FPP will allow the same.

## 3.3 Fixed form Fortran

### 3.3.1 A # sign in column 6 is not recognized as introducing a directive line.

| ID | Status | Sources |
|---|---|---|
| [fortran-cont-fixed-6-not-directive] | TBD | che1 |

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

Rationale: The corpus of example programs has instances where column 6 contains `#`, but only to mark continuation lines. The corpus does not have an example with `#` in column 6 for a preprocessor directive.

### 3.3.2 Identifier tokens are not broken by fixed-form line continuations

| ID | Status | Sources |
|---|---|---|
| [identifiers-span-continuations] | TBD | [F2023§6.2.2C601], jor4 |

**Tests** [Flpp pp005.F KWM split across continuation, implicit padding], [Flpp pp006.F ditto, but with intervening *comment line]

Rationale: 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.

## 3.4 Free form Fortran

### 3.4.1 Remove Fortran & line continuation in free-form Fortran text

| ID | Status | Sources |
|---|---|---|
| [fortran-cont-free-amp] | TBD | che1, [C§6.10 Preprocessing directives] |

Rationale: 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].

### 3.4.2 Remove Fortran `&` line continuation followed by `!` comment in free-form Fortran text

| ID | Status | Sources |
|---|---|---|
| [fortran-cont-free-amp-comment] | TBD | che1,<br>[C§6.10 Preprocessing directives],<br>[F2023§6.3.2.3 Free form commentary],<br>[F2023§6.3.2.4 Free form statement continuation] |

Rationale: 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].

### 3.4.3 Remove leading spaces before `&` in Fortran line continuation in free form

| ID | Status | Sources |
|---|---|---|
| [fortran-cont-free-rm-lead-amp] | TBD | che1, [C§6.10 Preprocessing directives] |

Rationale: 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.

## 3.5 Questions

### 3.5.1 How do we handle comments embedded between continuation lines?

### 3.5.2 Are any comment-based directives sandwiched between continuation lines?

# 4 Syntax

The following extended BNF grammar describes the lancuace recognized by Phase 0.

```
; Fortran file syntax for the Line Conjoiner

; The input to the Line Conjoiner is a sequence of
; fixed-form Fortran lines or directives
```

```
; or a sequence of free-form Fortran lines and directives.
; A file contains either fixed-form or free-form lines, not both.

InputFile = { FortranFixedSourceLine | DirectiveLine }*
          | { FortranFreeFormSourceLine | DirectiveLine }* .

;;;; Fixed Form

; A fixed-source Fortran line is either a comment or
; something that looks like a Fortran statement.
FortranFixedSourceLine = FixedCommentLine | FixedFortranStatementLine .

; Fortran fixed-form comment lines begin with a "C" or "c"
; or "*" in column 1. They can also begin with arbitrary whitespace
; followed by "!". They can contain any character available.
; Note that we explicitly call out the end-of-line as the
; special ?EOL? marker. The Fortran standard does not
; explicitly state how lines end (e.g., what characters represent /new-line/).
FixedCommentLine = ( "C" | "c" | "*" ) AnyChar ?EOL?
                 | OptWS "!" AnyChar ?EOL? .

; A Fortran fixed-form statement contains an optional line number
; in columns 1-5, a space in column 6, and the Fortran statement text.
; This initial line can be followed by continuation lines, or comments that
; might be interspersed between the continuation lines.
FixedFortranStatement = FixedFortranLineStart .

FixedFortranLineLstart = FixedContinuationLine FixedContinuationLineMess .

FixedContinuationLineMess = (FixedCommentLine* FixedContinuationLine)* .

FixedContinuationLine = FixedStatementNumber FortranChar* ?EOL? .

; Syntactically, a fixed-form statement number is any set of digits in
; column 1-5, ignoring whitespace in determining the statement label.
; Semantically, the number zero is not allowed to be a statement number.
; The Conjoiner doesn't care.
FixedStatementNumber = 5*5 (WS | "[0-9]") .
```

```
;;;; Free-Form

; Free-form source lines contain an optional statement number
; followed by statement text. Comments can appear up to the end of the line.
; We have to be a bit more careful here that in fixed-form, as there
; may be strings that we have to scan correctly to avoid misinterpreting comments.
; Continuation Lines end in "&", which may be followed by a comment.
; Continued lines may begin with whitespace followed by "&", which is
; stripped by the Conjoiner.
FortranFreeFormSourceLine = [ StatementNumber ] FreeSourceFragment [ FreeComment ] ?EOL
                          | [ StatementNumber ] ( FreeSourceFragment "&" [ Comment ] ?EOL

StatementNumber = 1*5 Digit .
FreeSourceFragment = ( FortranString | FortranChar - "[&'\"]" )* .

; Fortran strings are delimited with either single of double quotes.
; Doubled delimiters are allowed, representing a single delimiter.
FortranString = "\"" ((AnyChar - "\"") | "\"\"") * "\""
              | "'"  ((AnyChar - "'")  | "''" )* "'" .



;;;; Directives

; A directive begins with a "#" preceded by optional whitespace.
; and continues through escaped new-line characters until
; a line with a non-escaped new-line character.
DirectiveLine = optWS "#" DirectiveMaybeContinued .
DirectiveMaybeContinued = { FortranChar* "\\" ?EOL? }*  DirectiveFinalLine
                        | { FortranChar - "/" "[^*]" FortranChar  "\\" ?EOL? }*
                        | { FortranChar - "/*" (AnyChar + ?EOL?) "*/" FortranChar "\\"
DirectiveFinalLine = FortranChar* ?EOL? .

; The Fortran character set is any whitespace, alphanumeric character,
; or one of the special symbols below. [F2023\S6.1 Processor character set].
; This applies to both fixed-form and free-form source.
FortranChar = WS | AlphaNumericChar | "[=+-*/()[]{},.:;!%&~<>?'`^|$#@]" .
AlphaNumericChar = "[A-Za-z_]" | DigitChar .
DigitChar = "[0-9]" .
```

```
; Whitespace in Fortran is either a space character or tab character.
WS = ( ?space? | ?tab? ) .

; Optional whitespace is zero or more occurrences of whitespace.
OptWS = WS* .

; Some contexts allow any character. We represent this by
; the characters of extended ASCII. The Fortran standard
; does not mention anything like Unicode characters.
; Should revisit this to exclude newlines and such.
AnyChar = "[\000-\377]" - "[\008\009\012]".
```

## References


[ISO18] ISO/IEC. Information technology – programming languages – C. Standard ISO/IEC 9899:2018, International Organization for Standardization, Geneva, CH, July 2018.

[ISO23] ISO/IEC JTC 1. Information technology – programming languages – Fortran. Standard ISO/IEC 1539-1:2023, International Organization for Standardization, Geneva, CH, November 2023.

[ISO24] ISO/IEC. Information technology – programming languages – C. Standard ISO/IEC 9899:2018 (R2024), International Organization for Standardization, Geneva, CH, October 2024.

[SW24] Richard M. Stallman and Zachary Weinberg. The C preprocessor, 2024.