FPP Phase 4: Preprocess Directives

INCITS/Fortran JoR

December 10, 2024

Contents

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 4 of the preprocessor, which processes directives after Phase 3 tokenizes the input stream.

2 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.

2.1 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 `#`).

2.2 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.

2.3 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.

2.4 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.

3 High-level requirements

Steve Lionel, the ISO WG 5 (Fortran) Convenor, presented this list of high-level requirements at INCITS/Fortran meeting #232, February 2024 [Sub24]

Fortran programmers use preprocessing for several reasons:

- Adjusting external names for C interoperability (largely obviated by C interop features in the language)
- Platform/OS specific code
- Debug or other variants of code
- A crude way of implementing templates

Explicit goals for Fortran 202y were

- Define cpp-style preprocessor in the standard
- Ideally, most existing uses of preprocessing will "just work", or need minimal changes
- Feature will not be optional, but implementations are encouraged to offer an option to "do it the old way"
- "Minimum Viable Product" – don't try to do everything

Features should have the same semantics as the C preprocessor, except as noted.

- `__LINE__` and `__FILE__` defined macros
- `#line`
- `#ifdef`, `#ifndef`, `#endif`
- `#if`, `#elif`, `#else`, `#endif`
- `#define` and `#undef`
- `#include`
- `#error`

- `#` operator (character literal from token)
- `##` operator (token concatenation)
- `/* ... */` C-style comments are allowed in directives (including multi-line comments)
- \ new-line continuations allowed in directives
- `#pragma` directive for implementation-specific directives
- Macro expansion
  - tokens are case-sensitive
  - tokens are not replaced in character literals
  - tokens are not replaced in Hollerith strings
  - tokens are not replaced in *letter-spec-list* of `IMPLICIT` statements
  - tokens are not replaced in column 6 in fixed-form source
  - tokens are replaced in comment-style directives (such as `!omp$`, `!dir$`)
- Expressions
  - `defined` operator
  - `//` is the Fortran concatenation operator; it does not introduce a C-style comment
  - `!` is the C "not" operator; it does not introduce a Fortran comment

4 Detailed requirements

We list the detailed requirements for handling preprocessor directives.

Each requirement is a heading with a "One-line description", followed by a short table that contains a unique identifier, the requirement's current status in the INCITS/Fortran approval process, and where the requirement came from (such as a C or Fortran standard, an INCITS meeting, or on email conversation).

[Note: We omit the detailed requirements; I postponed work on this to focus on semantics.]

5 Syntax

The Bison grammar recognized in Phase 4 is in a separate file. [DS21]

References

[DS21] Charles. Donnelly and Richard M. Stallman. *Bison Manual: Using the YACC-Compatible Parser Generator, for Version 1. 875*. Free Software Foundation, 2021.

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

[Sub24] INCITS/Fortran JoR Subgroup. Preprocessor, take 2, Feb 2024.