Grammar Start Symbol

Randy Meyers

N695 J11/97-058
May 27, 1997

1. Introduction

Clive *et al*. in N579 identified several problems with the term "translation unit" and pointed out the start symbol for the grammar was not clearly indicated.

The *start symbol* of a grammar is the rule that recognizes sentences in the grammar. In the grammar for a programming language, the start symbol is usually the rule for program or translation unit. In C, there are two start symbols.

The first start symbol is *preprocessor-file* defined in Section 6.8, which is a rule describing the form of each source file or header processed during the preprocessing phases of translation. Loosely speaking, this grammar accepts preprocessor token followed by preprocessor token intermixed with preprocessor directives.

The second start symbol is *translation-unit* defined in Section 6.7, which is a rule describing the fully preprocessed source at translation phase 7 (traditional syntactic and semantic analysis). A *translation-unit* is a sequence of one or more external definitions.

That said, there are several problems in the Standard.

Section 5.1.1.1 gives an English definition of translation unit as "A source file together with all the headers and source files included via the preprocessing directive #include, less any source lines skipped by any of the conditional inclusion preprocessing directives." This seems to imply by omission that macros have not been expanded, but you can not process **`#if`** directives (the expression might reference a macro) or **`#include`** directives (the header name might be a macro) without doing macro expansion. Furthermore, the phases of translation do not justify not interleaving the processing of macro expansion and other directives.

In some places describing preprocessing, the Standard makes reference to the translation unit, but the translation unit (at least as defined by the grammar) only exists after all preprocessing is done. For example, "each argument's preprocessing tokens are completely macro replaced as if they formed the rest of the translation unit" would seem to mean that no further macros are replaced.

2. Wording Changes

The following proposed wording changes attempt to remove these problems and allow "translation unit" to unambiguously mean what the grammar rule says it means.

2.1 Change to 5.1.1.1 Program structure

Change:

> The text of the program is kept in units called source files in this International Standard. A source file together with all the headers and source files included via the preprocessing directive #include, less any source lines skipped by any of the conditional inclusion preprocessing directives, is called a translation unit.

to:

The text of the program is kept in units called source files, also known as preprocessing files, in this International Standard. A source file together with all the headers and source files included via the preprocessing directive #include is known as a preprocessing translation unit. After preprocessing, a preprocessing translation unit is called a translation unit.

Add to Forward References:

External definitions (6.7), Preprocessing directives (6.8)

Reason for changes:

By equating source file to preprocessor file, we establish a correspondence between source files and the start symbol in the grammar. By clearly stating that a translation unit has undergone full preprocessing, we become more compatible with the grammar for translation unit. Section 6.7 gives the grammar for translation unit. Section 6.8 gives the grammar for preprocessing file and defines "preprocessing" in paragraph 5.

2.2 Change to 5.1.1.2 Translation phases

Change:

7. White-space characters separating tokens are no longer significant. Each preprocessing token is converted into a token. The resulting tokens are syntactically and semantically analyzed and translated.

to:

7. White-space characters separating tokens are no longer significant. Each preprocessing token is converted into a token. The resulting tokens are syntactically and semantically analyzed and translated as a translation unit.

Add to Forward References:

External definitions (6.7)

Reason for change:

This identifies "translation unit" as the start symbol and forward references the definition of grammar rule.

2.3 Change to 5.1.1.3 Diagnostics

Change:

A conforming implementation shall produce at least one diagnostic message (identified in an implementation-defined manner) for every translation unit that contains a violation of any syntax rule or constraint, even if the behavior is also explicitly specified as undefined or implementation-defined.

To:

A conforming implementation shall produce at least one diagnostic message (identified in an implementation-defined manner) if a preprocessing translation unit or translation unit contains a violation of any syntax rule or constraint, even if the behavior is also explicitly specified as undefined or implementation-defined.

Reason for change:

A violation of syntax or constraint that occurs during preprocessing should produce a diagnostic. For example:

```
#if 10 ==
```

2.4 Change to 5.2.4.1 Translation limits

Change:

> 4095 macro identifiers simultaneously defined in one translation unit

to:

> 4095 macro identifiers simultaneously defined in one preprocessing translation unit

Reason for change:

Translation units only exist after preprocessing has completed. Note that the word “simultaneously," I believe, makes clear that definitions in failing **`#if`** clauses do not count towards the limit.

2.5 Change to 6.8.3.1 Argument substitution

Change:

> Before being substituted, each argument's preprocessing tokens are completely macro replaced as if they formed the rest of the translation unit; no other preprocessing tokens are available.

To:

> Before being substituted, each argument's preprocessing tokens are completely macro replaced as if they formed the rest of the preprocessing file; no other preprocessing tokens are available.

Reason for change:

Translation units only exist after preprocessing, and so the original text accidentally implies no macro substitution.

2.6 Change to 6.8.3.5 Scope of macro definitions

Possible change:

> A macro definition lasts (independent of block structure) until a corresponding #undef directive is encountered or (if none is encountered) until the end of the translation unit.

to:

> A macro definition lasts (independent of block structure) until a corresponding #undef directive is encountered or (if none is encountered) until the end of the translation phase 4.

Reason for change:

While it is true that macro definitions do not persist after the end of the translation unit, it is also true that macro definitions need not exist past the end of preprocessing in phase 4. Thus, this change is not strictly needed, but avoids the awkwardness of describing macro definitions in terms of the translation unit, which only exists after all of the macro expansions have been completed and the need for macro definitions has passed.

2.7 Change to 7.1.2 Standard Headers

At end of :

> [Footnote] 112. A header is not necessarily a source file, nor are the < and > delimited sequences in header names necessarily valid source file names.

add a new sentence:

> Syntactically, the contents of a header act as if it was a preprocessing file.

Reason for change:

This change is optional: currently, we say nothing about the form of contents of a header file, and the committee might wish to continue this silence. The above change would have the effect of outlawing header files that end in a partial preprocessor directive, etc., but since it is only an “as if” rule, it does not prohibit precompiled headers.

Note that footnote 112 states some fairly importance principles that cannot be deduced from the normative text (except by the absence of any specification to the contrary). Perhaps footnote 112 should be moved into normative text.