Standard Pragmas – Improved Wording
Randy Meyers
N696 J11/97-059
May 27, 1997

1. Introduction

This paper describes the problem of macro replacement in pragmas and suggests words.

2. Macro Replacement in Pragmas

In the C89 Standard, not much is said about the **`#pragma`** directive beyond that it causes the implementation to behave in an implementation defined manner. In particular, the Standard does not say anything about whether macros are replaced in the *pp-tokens* in the directive.

DEC's compiler took the approach that macros were expanded for consistency with **`#include`**, **`#if`**, **`#elif`**, and **`#line`**, all of which are defined by the Standard to expand macros. This turned out to invade the user's namespace.

For example, DEC C has a pragma named **`#pragma message`** that allows the user to control whether a diagnostic will be issued by the compiler. The directive **`#pragma message save`** will push the current state of enabled and disabled messages on a stack, so the error message state can be restored at a later time. Message control turned out to be very useful to the implementers as well as the users. For example, our system header files would not trigger any diagnostics even at the highest levels of flagging because pragmas were used to disable messages about portability and extensions (when users request the compiler to check their programs for portability, they are not interested if finding out the system headers are not portable).

For example, **`<stdio.h>`** might use **`#pragma message save`** at the start of the header, disable the optional portability message about bitfields of more that 16 bits (which are not portable to systems with 16 bit ints), and then restore the message state at the end of the header (just in case the user had enabled the message about bitfields wider than 16 bits).

This was a great idea until the first user defined a macro **`save`** before including **`<stdio.h>`**. By allowing macro expansions in pragma lines, we had stolen all of the pragma specific keywords from the user's namespace. DEC C solved this problem by allowing any keyword in a pragma to be prefixed by two underscores. Thus, **`<stdio.h>`** now begins with **`#pragma __message __save`**. Since the new keywords all begin with two underscores, they are in a part of the namespace reserved from users.

Newer pragmas in DEC C are defined not to permit macro expansions. For example, **`#pragma environment`** does not macro expand. The preprocessor looks at the token following **`#pragma`**, and if it is **`environment`**, then the compiler does not macro expand **`environment`** or any other token on the line.

I recommend that any new standard pragmas (such as those from N631, approved at Hawaii) avoid problems by specifying that macros are not expanded in the standard pragmas.

It appears that the new **`pragma()`** operator avoids the macro expansion problem:

> The resulting sequence of characters is processed through translation phase 3 to produce preprocessing tokens that are executed as if they were the pp-tokens in a pragma directive. [Section 6.8.9]

Macro replacement does not occur until phase 4. By only preprocessing pragma operator tokens through Phase 3, macro expansion is avoided.

However, there is still a problem with the **`#pragma`** directive. I recommend that Section 6.8.6:

> 6.8.6 Pragma directive
>
> Semantics
>
> A preprocessing directive of the form
>
> **`# pragma`** *pp-tokens*$_{opt}$ *new-line*
>
> causes the implementation to behave in an implementation-defined manner. Any pragma that is not recognized by the implementation is ignored.

be rewritten as follows:

> 6.8.6 Pragma directive
>
> Semantics
>
> A preprocessing directive of the form
>
> **`# pragma`** *pp-tokens*$_{opt}$ *new-line*
>
> where the preprocessing token **`STDC`** does not follow the **`pragma`** on the directive causes the implementation to behave in an implementation-defined manner. Any such pragma that is not recognized by the implementation is ignored.
>
> If the preprocessing token **`STDC`** does follow the **`pragma`** on the directive, then no macro replacements are performed on the directive, and the directive shall have one of the following forms whose meaning is described elsewhere:
>
> **`#pragma STDC FP_CONTRACT`** *on-off-switch*
> **`#pragma STDC FENV_ACCESS`** *on-off-switch*
> **`#pragma STDC CX_LIMITED_RANGE`** *on-off-switch*
>
> *on-off-switch*: one of:
> **`ON     OFF     DEFAULT`**

There should also be a forward reference to the section describing the standard pragmas.

The above wording incorporates and supersedes the wording changes to Section 6.8.6 in N631.