The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.4.8 Preprocessing numbers

pp-number syntax

pp-number:
digit
. digit
pp-number digit
pp-number identifier-nondigit
pp-number **e** *sign*
pp-number **E** *sign*
pp-number **p** *sign*
pp-number **P** *sign*
pp-number **.**

Commentary

This syntax supports the creation of preprocessing tokens that have no meaning as integer or floating-point tokens. The rationale for this relaxed syntax was to simplify the lexing of characters into preprocessing tokens rather than to support the stringizing of rather unusual sequences of characters. This flexibility also allows constructs such as:

```
1  #define glue(a, b) a ## b
2
3  float f = glue(1e+, 20);
```

to work as expected. The *identifier-nondigit* is needed to support hexadecimal constants.

There have been a number of requests to WG14 to tighten up the syntax of *pp-number*. The stated aim being to reduce the number of cases where a sequence of characters are treated as a *pp-number*, but cannot be converted to the token constant. The Committee's response to these requests is given in DR #003.

C90

The C90 Standard used the syntax nonterminal *nondigit* rather than *identifier-nondigit*.

Rationale

C99 replaces *nondigit* with *identifier-nondigit* in the grammar to allow the token pasting operator, ##, to work as expected. Given the code

```
#define  mkident(s)  s ## 1m
/* ... */
int mkident(int) = 0;
```

if an identifier is passed to the `mkident` macro, then 1m is parsed as a single pp-number, a valid single identifier is produced by the ## operator, and nothing harmful happens. But consider a similar construction that might appear using Greek script:

```
#define  μμμμk(p)  p ## 1μ
/* ... */
int μk(int) = 0;
```

For this code to work, 1μ must be parsed as only one pp-token. Restricting pp-numbers to only the basic letters would break this.

Support for additional digits via UCNs is new in C99. Also support for *p* and *P* in a *pp-number* is new in C99.

C++

Support for *p* and *P* in a *pp-number* is new in C99 and is not specified in the C++ Standard.

Other Languages

Most languages restrict the sequence of characters that can occur in an integer or floating-point token to those that are meaningful numbers. But then most languages do not include a preprocessor. The *p* and *P* form of exponents is unique to C.

Example

The character sequences:

```
1   22 .7  3.1E-41 1.2.3  4E+5SOME_OBJ6e+7
```

form the *pp-numbers*:

```
{22} {.7}  {3.1E-41} {1.2.3}  {4E+5SOME_OBJ6e+7}
```

not:

```
{22} {.7}  {3.1E-41} {1.2} {.3}  {4E+5} {SOME_OBJ} {6e+7}
```

Description

928 A preprocessing number begins with a digit optionally preceded by a period (`.`) and may be followed by valid identifier characters and the character sequences **e+**, **e-**, **E+**, **E-**, **p+**, **p-**, **P+**, or **P-**.

Commentary

This is a restatement of information given in the Syntax clause.

C90

Support for the *P* form of exponent is new in C99.

C++

The C++ Standard does not make this observation and like C90 does not support the *P* form of the exponent.

929 Preprocessing number tokens lexically include all floating and integer constant tokens.

Commentary

This describes in words what is specified in the syntax.

C++

This observation is not made in the C++ Standard.

Common Implementations

They also lexically include the binary constants supported by some implementations (e.g., `0b01010101`).

Semantics

930 A preprocessing number does not have type or a value;

Commentary

A preprocessing number is nothing more than a sequence of characters.

Other Languages

Languages that do not include a preprocessor usually give a value and a type to such tokens as soon as they are created, that is immediately after lexing.

it acquires both after a successful conversion (as part of translation phase 7) to a floating constant token or an integer constant token. 931

Commentary

translation phase 7
#if operand type uintmax_*

A preprocessing number may acquire a type and a value prior to translation phase 7 if it occurs within a **#if** preprocessor directive.

Coding Guidelines

#if operand type uintmax_*

A *pp-number* that occurs within a **#if** preprocessor directive is likely to have a different type than the one it would be given as part of translation phase 7. The implications of this difference are discussed elsewhere.

footnote 69

69) Thus, sequences of characters that resemble escape sequences cause undefined behavior. 932

Commentary

escape sequence syntax
translation phase 5

Escape sequences start with the character \, one of the characters whose appearance in a *header-name* causes undefined behavior. Developers may make an association between a sequence of characters starting with \ and escape sequences (which are not converted until translation phase 5, but at which time any *header-name* preprocessing token will have ceased to exist), which technically does not exist at the same time that the *header-name* preprocessing directive exists.

Common Implementations

On some translator host environments, in particular the MS-DOS file system, the \ character is the directory separator. Because of the large volume of existing source code using the MS-DOS directory naming conventions, many implementations also support it as a directory separator (treating it as equivalent to the separator actually used— e.g., / under Linux).

Example

```
1   #include "dir\phile.h"
```

footnote DR324

DR324) For an example of a header name preprocessing token used in a **#pragma** directive, see Subclause 6.10.9. 933

Commentary

_Pragma operator

This footnote was added by the response to DR #324.

References