Minor Changes to C9*x*

Douglas A. Gwyn

US Army Research Laboratory
Adelphi, MD

ABSTRACT

While looking things up in C9*x* Draft 10, I found several places where small improvements were needed. Some of these could be considered editorial, but others are clearly substantive. Most of the ideas for substantive changes arose in the course of editorial work on already-approved proposals.

Because we are trying to complete the C9*x* document as soon as possible, I propose only changes that can be quickly and easily made as part of the ongoing editing process. My intention is that the Menlo Park agenda include a session wherein each item would be briefly presented, questions would be answered, and a quick yes/no vote would be taken for each proposed change. (If any item were to require extended debate, then I would simply withdraw that proposal.)

1. keywords cannot be identifiers

1.1 the problem

There is a **Constraint** in **6.1.2 Identifiers** that says:

> In translation phases 7 and 8, an identifier shall not consist of the same sequence of characters as a keyword.

This injunction is already given in the preceding **6.1.1 Keywords**, **Semantics**, and I'm not sure that there is a reliable way to diagnose all violations.

1.2 my proposal

I suggest any of three alternatives:

1.2.1 Change the **Constraints** given in **6.1.2 Identifiers** to a footnote.

1.2.2 Move the **Constraints** given in **6.1.2 Identifiers** to the **Description** section of the same subclause.

1.2.3 Delete the **Constraints** given in **6.1.2 Identifiers**.

2. pure binary numeration system

2.1 the problem

The requirement that integer types be represented using a ‘‘pure binary numeration system’’ has important ramifications; however, the explanation of this phrase is delegated to a footnote, leading many readers of the Standard to believe that it is not normative.

2.2 my proposal

Move the text of Footnote 25 from **6.1.2.5 Types** to clause **3 Definitions and conventions** (new subclause **3.16+ Pure binary numeration system**), and delete the footnote reference mark ‘‘[25]’’.

3. string literal confusion

3.1 the problem

In the course of figuring out whether the requirement in **6.5.7 Initialization**, **Constraints** that each expression in an initializer list for certain objects be a constant expression properly allows for the use of string literals to initialize character arrays, I discovered some sloppiness in the use of the term ‘‘string literal’’. (Incidentally, the answer to the above question is ‘‘yes’’, although it depends on some accidents working out right.) **6.1.4 String literals** says that ‘‘A character string literal is a sequence of zero or more multibyte characters enclosed in double-quotes’’, and similarly for wide string literals. However, for the logic of **6.5.7 Initialization** to work out right, ‘‘string literal’’ really needs to designate the resulting object after translation phase 7. The **Example** in **6.1.4 String literals** is also phrased in such a way as to indicate this. (The **Example** is incorrect in stating that there are two characters contained in the resulting string literal; there are actually *three*, including the terminating null byte.)

3.2 discussion

It is tricky to fix any of this without breaking things. Therefore I propose a very slight tweak to the existing wording.

3.3 my proposal

In **6.1.4 String literals**, **Description**, change the first sentence from

> A character string literal is a sequence of zero or more multibyte characters enclosed in double-quotes, as in `"xyz"`.

to

> A character string literal is designated by a sequence of zero or more multibyte characters enclosed in double-quotes, as in `"xyz"`.

In **6.1.4 String literals**, **Examples**, change

> produces a single character string literal containing the two characters whose values are `´\x12´` and `´3´`,

to

> produces a single character string literal containing the three characters whose values are `´\x12´`, `´3´`, and `0`,

4. strange characters in example

4.1 the problem

One of the **Examples** at the end of **6.1.7**, which doesn’t belong in this subclause anyway, uses characters that are not in the basic source character set, rendering the example not strictly conforming.

4.2 my proposal

In **6.1.7 Header names**, **Examples**, delete the two lines:

```
#define const.member@$
```

and

```
{#}{define} {const}{.}{member}{@}{$}
```

5. errors in example comments

5.1 the problem

A few typographic errors crept into the **Examples** in **6.1.9 Comments**.

Also, the four-character constant is unnecessary and perhaps even unwise, since it is not strictly conforming code, and the four-character string literal illustrates the same point.

5.2 my proposal

Replace **6.1.9 Comments**, **Examples** content with the following:

```
"a//b"            // four-character string literal
#include "//e"    // undefined behavior
// */             // comment, not syntax error
f = g/**//h;      // equivalent to f = g / h;
//\
i();              // part of two-line comment
/\
/ j();            // part of two-line comment
```

```
#define glue(x,y) x##y
glue(/,/) k();   // syntax error, not comment
/*//*/ l();      // equivalent to l();
m = n//**/o
  + p;           // equivalent to m = n + p;
```

6. pointer ←→ integer conversion

6.1 the problem

Apparently DR #57 said that there doesn't have to exist an integer type suitable for holding any object pointer, but the wording in the C9x draft about this, which has not changed since C89, can easily be read as stating the opposite.

A second problem is that the current wording requires the implementation to define the meaning for every conversion of an arbitrary integer value to a pointer, which seems to impose an unnecessarily heavy burden on some implementations.

6.2 my intention

I agree that it is best not to require that implementations provide such a type; also that it can be useful for implementations to support such conversions. Because the current wording does not guarantee that the type must exist, and the ‘‘requirement’’ occurs in a **Semantics** section, I believe that the following change would not affect any implementation or program, but it would more clearly indicate the nature of such a facility. (It could make implementation conformance easier in some cases.)

6.3 my proposal

Remove the following words from **6.3.4 Cast operators**, **Semantics**:

> A pointer may be converted to an integer type. The size of integer required and the result are implementation-defined. If the space provided is not long enough, the behavior is undefined.
>
> An arbitrary integer may be converted to a pointer. The result is implementation-defined.[58]

along with Footnote 58.

In **I.5 Common extensions**, add a new subclause **I.5.6+ Conversion between pointers and integers**:

> A pointer may be converted to some integer type, and some integers may be converted to pointers. Details of these conversions are implementation-defined. (6.3.4)

7. trailing comma on enumerator list

7.1 the problem

Many people, including Ken Thompson, responded unfavorably to C89's requirement that a trailing comma after an enumerator list must be diagnosed, contrary to previous widespread existing practice. This requirement is inconsistent with C89's treatment of trailing comma after an initializer list (it's optional there), and it complicates programs that generate C source code.

7.2 my proposal

In **6.5.2.2 Enumeration specifiers**, **Syntax**, add another derivation rule to the grammar for
enum-specifier:

`enum` *identifier*$_{opt}$ `{` *enumerator−list* `, }`

Also make the corresponding change in **B.2.2 Declarations**.

8. erroneous footnote 82

8.1 the problem

Footnote 82 erroneously says that only through the use of typedefs can array or function type specifications include any type qualifiers.

8.2 my opinion

I think this footnote was trying to be helpful, but incorrect footnotes are not helpful.

8.3 my proposal

In **6.5.3 Type qualifiers**, delete Footnote 82 and the reference to it.

9. typedefs don’t define

9.1 the problem

Some people have found the terminology of **6.5.6 Type definitions** misleading.

9.2 my proposal

Change the name to **6.5.6 Typedefs**, and in the first sentence of **Semantics** change

> each declarator defines

to

> each declarator declares

10. _Pragma missing from grammar

10.1 the problem

There ought to be a derivation rule involving the new `_Pragma` operator.

10.2 my proposal

In **6.8 Preprocessing directives**, **Syntax**, add another derivation rule for
control-line:

> `_Pragma (` *string-literal* `)`

11. misplaced macro-replacement constraint

11.1 the problem

The last **Constraint** in **6.8.1 Conditional inclusion** has nothing to do with conditional inclusion.

11.2 my proposal

Move the last paragraph from **6.8.1 Conditional inclusion**, **Constraints**:

> Each preprocessing token that remains after all macro replacements have occurred shall be in the lexical form of a token.

to a new **Constraints** section at the beginning of **6.8.3.4 Rescanning and further replacement**, and follow this with the (previously implicit) section heading **Semantics**.

12. replacement of object-like macro identifier

12.1 the problem

The macro replacement semantics incorrectly include among identifiers that are replaced by macro expansion those that are used as macro names in various preprocessing directives.

12.2 my proposal

In **6.8.3 Macro replacement**, **Semantics**, change

> defines an object-like macro that causes each subsequent instance of the macro name[106]

to

> defines an object-like macro that causes each subsequent instance of the macro name[106], except for the identifier in another `#define`, `#undef`, `#ifdef`, or `#ifndef` directive or as the operand of a `defined` operator,

13. type qualifiers in standard headers

13.1 the problem

An implementation may declare typedef names as qualified types in the standard headers. This can cause unpleasant surprises.

13.2 note

This is *not* just nit-picking; I have heard that at least one implementation already includes `volatile` in its declaration of `sig_atomic_t`.

13.3 my proposal

I actually have two alternative formulations, depending on whether there is a consensus that some standard typedef names (*e.g.,* `sig_atomic_t`) ought to be allowed to be declared as qualified types.

13.3.1 At the end of **7.1.2 Standard headers** add a new paragraph:

> Declarations of types described in this clause shall not include type qualifiers.

13.3.2 At the end of **7.1.2 Standard headers** add a new paragraph:

> Declarations of types described in this clause shall not include type qualifiers, unless explicitly stated otherwise.

13.3.2.1 If it is decided that a contrary type is `sig_atomic_t`, then also in **7.10 Signal handling** `<signal.h>` change

> which is the integer type of an object

to

> which is the (possibly volatile-qualified) integer type of an object

14. math error conditions misplaced

14.1 the problem

Subclause **7.7.3 Treatment of error conditions** is embedded in the middle of functional descriptions, instead of being near the front of the `<math.h>` subclause, which would be more in line with the general style of the Standard.

14.2 my proposal

Move the entire subclause (including heading, with automatic renumbering) **7.7.3 Treatment of error conditions** in front of **7.7.1 The `FP_CONTRACT` pragma**.

15. missing words for lgamma

15.1 the problem

The `lgamma` **Returns** specification is inaccurate.

15.2 my proposal

In **7.7.8.4 The `lgamma` function**, **Returns**, change ‘‘gamma’’ to ‘‘gamma of `x’’.`

16. race condition in signal handler

16.1 the problem

The C89 condition that a signal handler upon entry can have `SIG_DFL` established for the signal (which forces an unsafe timing window for asynchronous signals) is a hold-over from pre-POSIX days that encourages programmers to use less-standard signal mechanisms. (I argued for this specification originally, on the grounds of backward compatibility.)

A second, related problem is that too little is said about the implementation-defined blocking, making it difficult to write useful portable programs using `signal`.

16.2 my intention

The C89 alternative, implementation-defined blocking of the signal, should now be required. Just enough additional requirements should be imposed to guarantee a reasonable way to safely write portable programs that catch asynchronous signals such as `SIGINT`. I believe my proposal for the latter is compatible with current implementations of signal blocking, but I would appreciate feedback on this matter.

16.3 my proposal

I propose two independent changes that can be decided separately:

16.3.1 In **7.10.1.1 The `signal` function**, **Description**, second paragraph, change

> When a signal occurs, if `func` points to a function, first the equivalent of `signal(sig, SIG_DFL);` is executed or an implementation-defined blocking of the signal is performed. (If the value of `sig` is `SIGILL`, whether the reset to `SIG_DFL` occurs is implementation-defined.)

to

> When a signal occurs, if `func` points to a function, first an implementation-defined blocking of the signal is performed.

16.3.2 After the third paragraph in **7.10.1.1 The `signal` function**, **Description**, add a new paragraph:

> While a signal is blocked, a call to the `signal` function with `sig` equal to that signal number shall cause the signal to be unblocked.

Also, append to the fourth paragraph (concerning program startup) in **7.10.1.1 The `signal` function**, **Description**:

> Some signals may be blocked upon program startup.

17. more signal returns undefined

17.1 the problem

Returning from a signal handler for `SIGILL` or `SIGSEGV` ought to explicitly cause undefined behavior (because that's what happens in actuality).

17.2 my proposal

In **7.10.1.1 The `signal` function**, **Description**, second paragraph, change

> and the value of `sig` was `SIGFPE`

to

> and the value of `sig` was `SIGFPE`, `SIGILL`, `SIGSEGV`,

18. functions in signal handler

18.1 the problem

The words ‘‘the `signal` function itself’’ might be confusing, since ‘‘itself’’ more properly refers to the signal handler.

18.2 my solution

In **7.10.1.1 The `signal` function**, **Description**, third paragraph, change

> other than the `signal` function itself

to

> other than the `signal` function

19. opening text file in update mode

19.1 the problem

The draft says,

> Opening (or creating) a text file with update mode may instead open (or create) a binary stream in some implementations.

This doesn't seem like a specification so much as a warning. Since an attempt to open a file inappropriately is allowed to fail, it is much better in such a situation for `fopen` to report failure than to secretly shift the mode of the stream (which could invalidate further processing).

19.2 my proposal

In **7.12.5.3 The `fopen` function**, **Description**, delete that sentence:

> Opening (or creating) a text file with update mode may instead open (or create) a binary stream in some implementations.

20. common fprintf extension

20.1 the problem

There is no good reason to disallow `%lf` in `fprintf` format strings. It makes sense, improves compatibility with `fscanf` format strings, and is commonly implemented as a conforming extension. Also, `%d` *vs.* `%hd` is supported, which makes no more or less sense than `%lf` *vs.* `%f`.

20.2 comment

I suspect that one reason this was left out was the difficulty in fitting it into the wording pattern of the existing text.

20.3 my proposal

In **7.12.6.1 The `fprintf` function**, **Description**, before

> or an optional `L` specifying

insert the following:

> an optional `l` followed by an `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier (the `l` has no effect);

In **7.18.1.1 The `fwprintf` function**, **Description**, before

> or an optional `L` specifying

insert the following:

> an optional `l` followed by an `a`, `A`, `e`, `E`, `f`, `F`, `g`, or `G` conversion specifier (the `l` has no effect);

21. errors in fscanf spec

21.1 the problem

`fscanf`'s `%i`, `%d`, and `%n` are said to have arguments that are merely pointers to integer; that should be *signed* integer.

Also, the undefined term ''valid'' is used, whereas the `fprintf` specification contains better wording that could be adopted for `fscanf`.

There is also a misleading ''which'' where the restrictive ''that'' is needed.

21.2 my proposal

The following three items can be decided separately.

21.2.1 In **7.12.6.2 The `fscanf` function**, **Description**, in the paragraph beginning ''An input item is read from the stream,'' change

> the longest sequence of input characters which does not exceed

to

> the longest sequence of input characters that does not exceed

In **7.18.2.2 The `fwscanf` function**, **Description**, in the paragraph beginning ''An input item is read from the stream,'' change

> the longest sequence of input wide characters which does not exceed

to

the longest sequence of input wide characters that does not exceed

21.2.2 In **7.12.6.2 The `fscanf` function**, **Description**, change

The following conversion specifiers are valid:

to

The conversion specifiers and their meanings are:

In **7.18.2.2 The `fwscanf` function**, **Description**, change

The following conversion specifiers are valid:

to

The conversion specifiers and their meanings are:

21.2.3 In **7.12.6.2 The `fscanf` function**, **Description**, under each of the conversion specifiers `d`, `i`, and `n`, change

shall be a pointer to integer

to

shall be a pointer to signed integer

In **7.18.2.2 The `fwscanf` function**, **Description**, under each of the conversion specifiers `d`, `i`, and `n`, change

shall be a pointer to integer

to

shall be a pointer to signed integer

22. numeric conversion for char type

22.1 the problem

The lack of a numeric conversion specification in `fscanf` for `char` data analogous to `%hd` for `short`, `%d` for `int`, `%ld` for `long`, and `%lld` for `long long` became quite apparent while I was attempting to update the `<inttypes.h>` specification (Document WG14/N761). Such a format conversion could be quite useful in reading Boolean values, for example.

There are also a couple of errors associated with the addition of wording for `ll`.

22.2 discussion

The only reason I can see for this omission is historical inertia.

To maintain maximum parallelism between `fprintf` and `fscanf`, we ought to add the facility to both families rather than just to `fscanf`.

I intentionally don’t try to cover `wchar_t` data, because that’s not one of the basic standard integer types.

22.3 my proposal

I suggest two alternatives for how to spell the conversion modifier, `hh` or `H`. Here I show just the `hh` alternative (which is the one I prefer); it would be easy enough to edit this to use another spelling.

22.3.1 In **7.12.6.1 The `fprintf` function**, **Description**, change

— An optional `h` specifying that a following `d`,

to

— An optional `hh` specifying that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a `signed char` or `unsigned char` argument (the argument will have been promoted according to the integral promotions, and its value shall be converted to `signed char` or `unsigned char` before printing); an optional `h` specifying that a following `d`,

In the same paragraph, change

If an `h`, `l`, or `L` appears with any other

to

If an `hh`, `h`, `l`, `ll`, or `L` appears with any other

In **7.12.6.2 The** `fscanf` **function**, **Description**, change

— An optional `h`, `l` (ell), or `ll` (ell-ell), or `L`

to

— An optional `hh`, `h`, `l` (ell), `ll` (ell-ell), or `L`

In the same paragraph, change

`d`, `i`, and `n` shall be preceded by `h` if the corresponding argument is a pointer to `short int` rather than a pointer to `int`,

to

`d`, `i`, and `n` shall be preceded by `hh` if the corresponding argument is a pointer to `char` rather than a pointer to `int`, by `h` if it is a pointer to `short int`,

In the same paragraph, change

`o`, `u`, and `x` shall be preceded by `h` if the corresponding argument is a pointer to `unsigned short int` rather than a pointer to `unsigned int`,

to

`o`, `u`, and `x` shall be preceded by `hh` if the corresponding argument is a pointer to `unsigned char` rather than a pointer to `unsigned int`, by `h` if it is a pointer to `unsigned short int`,

In the same paragraph, change

If an `h`, `l`, or `L` appears with any other

to

If an `hh`, `h`, `l`, `ll`, or `L` appears with any other

In **7.18.2.1 The** `fwprintf` **function**, **Description**, change

— an optional `h` specifying that a following `d`,

to

— an optional `hh` specifying that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a `signed char` or `unsigned char` argument (the argument will have been promoted according to the integral promotions, and its value shall be converted to `signed char` or `unsigned char` before printing); an optional `h` specifying that a following `d`,

In the same paragraph, change

If an `h`, `l`, `ll`, or `L` appears with any other

to

If an `hh`, `h`, `l`, `ll`, or `L` appears with any other

In **7.18.2.2 The** `fwscanf` **function**, **Description**, change

— An optional `h`, `l` (ell), `ll` (ell-ell), or `L`

to

— An optional `hh`, `h`, `l` (ell), `ll` (ell-ell), or `L`

In the same paragraph, change

`d`, `i`, and `n` shall be preceded by `h` if the corresponding argument is a pointer to `short int` rather than a pointer to `int`,

to

`d`, `i`, and `n` shall be preceded by `hh` if the corresponding argument is a pointer to `char` rather than a pointer to `int`, by `h` if it is a pointer to `short int`,

In the same paragraph, change

`o`, `u`, and `x` shall be preceded by `h` if the corresponding argument is a pointer to `unsigned short int` rather than a pointer to `unsigned int`,

to

> `o`, `u`, and `x` shall be preceded by `hh` if the corresponding argument is a pointer to `unsigned char` rather than a pointer to `unsigned int`, by `h` if it is a pointer to `unsigned short int`,

In the same paragraph, change

> If an `h`, `l`, `ll`, or `L` appears with any other

to

> If an `hh`, `h`, `l`, `ll`, or `L` appears with any other

23. seek on same file okay

23.1 the problem

Seek offsets are required to be obtained from the same *stream*, not just from the same *file.* That means one can't close a file and later reopen it and seek back into it.

23.2 discussion

It is *possible* that some operating system really imposes this limitation, but I don't recall any like that. My guess is that this was just imprecise wording that was never noticed.

In the following, I rely on the vagueness of the phrase ‘‘the same file’’ to handle cases such as when a file is somehow modified between the time of the tell and the seek; this technicality (and a few similar complications) should have been dealt with even in the previous formulation.

23.3 my proposal

In **7.12.10.2 The `fseek` function**, **Description**, change

> an earlier successful call to the `ftell` function on the same stream

to

> an earlier successful call to the `ftell` function on a stream associated with the same file

In **7.12.10.3 The `fsetpos` function**, **Description**, change

> an earlier successful call to the `fgetpos` function on the same stream.

to

> an earlier successful call to the `fgetpos` function on a stream associated with the same file.

24. use of an unfamiliar term

24.1 the problem

In the specification for `strtod`, what is a ‘‘binary-exponent part’’?

24.2 my proposal

In **7.13.1.5 The `strtod` function**, **Description**, change

> then an optional binary-exponent part,

to

> then an optional binary-exponent part as defined in 6.1.3.1,

25. missing reference to C89

25.1 the problem

Shouldn't there be a reference to our other base document, the previous C standard?

25.2 my proposal

In **Annex A Bibliography**, add the reference:

1. ISO/IEC 9899:1990, *Programming Languages — C*.

26. missing restrict in Annex B

26.1 the problem

`restrict` is missing from the grammar in **Annex B**.

26.2 my proposal

In **B.2.2 Declarations**, add under
(6.5.3) *type-qualifier:*
```
        restrict
```