N 601
Incorporate the `long long` integral data type into C9X

Roland Hartinger

Siemens Nixdorf Informationssysteme

ABSTRACT

This document describes the changes to the sixth C9X draft needed to introduce the new integral type **`long long`** into Standard C.

I. Document history

1. Changes since N572

- Clarified editorial instructions; moved Rationale parts to a separate document.
- Removed **`rinttoll`** and **`roundtoll`**; Jim Thomas will add them to the floating point C extension.

2. Changes since N499

- Removed the **`long long`** versions of **`fseek`** and **`ftell`**.
- Preprocessing arithmetic is now required to be performed as if all integers were **`long long`** or **`unsigned long long`**.
- In the description of **`fscanf`**, a recently added sentence that lists all possible specifications involving **`%n`** acquired a **`long long`** version.
- Added changes to **`fwscanf`** and **`fwprintf`**; they are almost identical to their non-**`wchar_t`** equivalents.
- Added **`rinttoll`** and **`roundtoll`** in anticipation of the floating point C extension's acceptance.
- Adapted to the changed section numbers of C9X draft 6.

II. Changes to the ISO/IEC 9899:1990, Programming Language C

The following sections describe the **`long long`** data type extension in terms of the ISO/IEC 9899:1990, Programming Language C Standard. Subclause numbers refer to the sixth C9X draft; where appropriate, changes have been underlined. Some new subclauses must be created to accommodate new **`long long`** library functions.

5 Environment

5.2.4.2.1 Sizes of integral types **`<limits.h>`**

Add the following macros to the end of paragraph 1:

— minimum value for an object of type **`long long int`**
`LLONG_MIN` **`-9223372036854775807`**

— maximum value for an object of type **`long long int`**
`LLONG_MAX` **`9223372036854775807`**

— maximum value for an object of type **`unsigned long long int`**
`ULLONG_MAX` **`18446744073709551615`**

6. Language

6.1.2.5 Types

Change the first sentence in paragraph 3 to:

> There are five *signed integer types*, designated as **`signed char`**, **`short int`**, **`int`**, **`long int`**, and **`long long int`**.

6.1.3.2 Integer constants

Extend the Syntax as follows:

integer-suffix:

unsigned-suffix long-long-suffix$_{opt}$
long-long-suffix unsigned-suffix$_{opt}$

long-long-suffix: one of

`ll LL`

Replace the second sentence of paragraph 6 with:

> Unsuffixed decimal: **`int`**, **`long int`**, **`unsigned long int`**, **`long long int`**, **`unsigned long long int`**; unsuffixed octal or hexadecimal: **`int`**, **`unsigned int`**, **`long int`**, **`unsigned long int`**, **`long long int`**, **`unsigned long long int`**; suffixed by the letter **`u`** or **`U`**: **`unsigned int`**, **`unsigned long int`**, **`unsigned long long int`**; suffixed by the letter **`l`** or **`L`**: **`long int`**, **`unsigned long int`**, **`long long int`**, **`unsigned long long int`**; suffixed by both the letters **`u`** or **`U`** and **`l`** or **`L`**: **`unsigned long int`**, **`unsigned long long int`**; suffixed by **`ll`** or **`LL`**: **`long long int`**, **`unsigned long long int`**; suffixed by both **`u`** or **`U`** and **`ll`** or **`LL`**: **`unsigned long long int`**.

6.2.1.5 Usual arithmetic conversions

Replace the first sentence of the indented section ending paragraph 1 with:

> If either operand has type **`unsigned long long int`**, the other operand is converted to **`unsigned long long int`**.
>
> Otherwise, if one operand has type **`long long int`** and the other has type **`unsigned long int`**, if a **`long long int`** can represent all values of an **`unsigned long int`**, the operand of type **`unsigned long int`** is converted to **`long long int`**; if a **`long long int`** cannot represent all the values of an **`unsigned long int`**, both operands are converted to **`unsigned long long int`**.
>
> Otherwise, if one operand has type **`long long int`** and the other has type **`unsigned int`**, if a **`long long int`** can represent all values of an **`unsigned int`**, the operand of type **`unsigned int`** is converted to **`long long int`**; if a **`long long int`** cannot represent all the values of an **`unsigned int`**, both operands are converted to **`unsigned long long int`**.
>
> Otherwise, if either operand is **`long long`**, the other operand is converted to **`long long`**.
>
> Otherwise, if either operand has type **`unsigned long int`**, the other operand is converted to **`unsigned long int`**.

6.3 Expressions

6.3.3.3 Unary arithmetic operators

Change the last two sentences of paragraph 5 to the following:

> The expression **~E** is equivalent to **(ULLONG_MAX−E)** if **E** is promoted to type **unsigned long long**, **(ULONG_MAX−E)** if **E** is promoted to type **unsigned long**, and to **(UINT_MAX−E)** if **E** is promoted to type **unsigned int**. (The constants **ULLONG_MAX** , **ULONG_MAX**, and **UINT_MAX** are defined in the header **<limits.h>**.)

6.3.7 Bitwise shift operators

Change the last two sentences of paragraph 5 to the following:

> If **E1** has an unsigned type, the value of the result is **E1** multiplied by the quantity, 2 raised to the power **E2**, reduced modulo **ULLONG_MAX+1** if **E1** has type **unsigned long long**, **ULONG_MAX+1** if **E1** has type **unsigned long**, **UINT_MAX+1** otherwise. (The constants f8ULLONG_MAX**, ULONG_MAX, and UINT_MAX are defined in the header <limits.h>.)**

6.5.2 Type specifiers

In paragraph 3, add the following two items to the list of type specifier sets in the Constraints paragraph, between **unsigned long** and **float**:

— **long long**, **signed long long**, **long long int**, or **signed long long int**

— **unsigned long long**, or **unsigned long long int**

6.8 Preprocessing directives

6.8.1 Conditional inclusion

In the middle of paragraph five, change

> The resulting tokens comprise the controlling constant expression which is evaluated according to the rules of 6.4 using arithmetic that has at least the ranges specified in 5.2.4.2, except that **int** and **unsigned int** act as if they have the same representation as, respectively, **long** and **unsigned long**.

to

> The resulting tokens comprise the controlling constant expression which is evaluated according to the rules of 6.4 using arithmetic that has at least the ranges specified in 5.2.4.2, except that **int** and **long**, and **unsigned int** and **unsigned long**, act as if they have the same representation as, respectively, **long long** and **unsigned long long.**

7. Library functions

7.10.6 Formatted input/output functions

7.10.6.1 The `fprintf` function

Change the text of the fourth item of paragraph 4 in part to read:

> ... **x**, or **X** conversion specifier applies to a **long int** or **unsigned long int** argument; an optional **ll** (ell-ell) specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long long int** or **unsigned long long int** argument; an optional **l** specifying that a following **n** conversion specifier applies to a pointer to a **long int** argument; an optional **ll** (ell-ell) specifying that a following **n** conversion specifier applies to a pointer to a **long long int** argument; an optional **l** specifying that the following **c** conversion ...

Change the same paragraph, same list item, last sentence:

> If an **h**, **l**, **ll**, or **L** appears with any other conversion specifier, the behavior is undefined.

7.10.6.2 The `fscanf` function

In paragraph 4, change the beginning of the third list item to:

> An optional **h**, **l** (ell), **ll** (ell-ell) or **L** indicating the size of the receiving object.

In the same list item, change the end of the second sentence:

> ..., or by **l** if it is a pointer to **long int**, or by **ll** if it is a pointer to **long long int**.

Similarly, change the end of the third sentence:

> ..., or by **l** if it is a pointer to **unsigned long int**, or by **ll** if it is a pointer to **unsigned long long int**.

Change the last sentence of the list item to:

> If an **h**, **l**, **ll**, or **L** appears with any other conversion specifier, the behavior is undefined.

In paragraph 15, change the last sentence of the item on **n** to

> If the conversion specification with this conversion specifier is not one of **%n**, **%ln**, **%lln**, or **%hn**, the behavior is undefined.

7.11 General utilities `<stdlib.h>`

Replace the first two paragraphs of 7.11 with:

> The header **<stdlib.h>** declares five types and several functions of general utility, and defines several macros.[140]
>
> The types declared are **size_t** and **wchar_t** (both described in 7.1.6),
>
> **div_t**
>
> which is a structure type that is the type of the value returned by the **div** function,
>
> **ldiv_t**
>
> which is a structure type that is the type of the value returned by the **ldiv** function, and
>
> **lldiv_t**
>
> which is a structure type that is the type of the value returned by the **lldiv** function.

7.11.1 String conversion functions

Change the first sentence of the first paragraph to:

> The functions **atof**, **atoi**, **atol** and **atoll** need not affect the value of the integer expression **errno** on an error.

Add the following three new subclauses:

7.11.1.7 The `atoll` function

Synopsis

```
#include <stdlib.h>
long long int atoll(const char *nptr);
```

Description

The **atoll** function is similar to the **atol** function, except that it converts the initial portion of the string pointed to by **nptr** to **long long int** representation. Except for the behavior on error, it is equivalent to

```
strtoll(nptr, (char **)NULL, 10)
```

Returns

The **atoll** function returns the converted value.

7.11.1.8 The `strtoll` function

Synopsis

```
#include <stdlib.h>
long long int strtoll(const char *nptr, char **endptr, int base);
```

Description

The **strtoll** function is similar to the **strtol** function, except that it converts the initial portion of the string pointed to by **nptr** to **long long int** representation.

Returns

The **strtoll** function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, **LLONG_MAX** or **LLONG_MIN** is returned (according of the sign of the value), and the value of the macro **ERANGE** is stored in **errno**.

7.11.1.9 The `strtoull` function

Synopsis

```
#include <stdlib.h>
unsigned long long int strtoull(const char *nptr, char ** endptr,
      int base);
```

Description

The **strtoull** function is similar to the **strtoul** function, except that it converts the initial portion of the string pointed to by **nptr** to **unsigned long long int** representation.

Returns

The `strtoull` function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `ULLONG_MAX` is returned, and the value of the macro `ERANGE` is stored in `errno`.

7.11.6 Integer arithmetic functions

Add the following two new subclauses:

7.11.6.5 The `llabs` function

Synopsis

```
#include <stdlib.h>
long long int llabs(long long int j);
```

Description

The `llabs` function is similar to the `abs` function, except that the argument and the returned value each have type `long long int`.

7.11.6.6 The `lldiv` function

Synopsis

```
#include <stdlib.h>
lldiv_t lldiv(long long int numer, long long int denom);
```

Description

The `lldiv` function is similar to the `div` function, except that the arguments and the members of the returned structure (which has type `lldiv_t`) all have type `long long int`.

7.17.2 Formatted wide-character input/output functions

7.17.2.1 The `fwprintf` function

Change the text of the fourth item of paragraph 5 in part to read:

> `long int` or `unsigned long int` argument; an optional `ll` (ell-ell) specifying that a following `d`, `i`, `o`, `u`, `x`, or `X` conversion specifier applies to a `long long int` or `unsigned long long int` argument; an optional `l` specifying that a following `n` conversion specifier applies to a pointer to a `long int` argument; an optional `ll` (ell-ell) specifying that a following `n` conversion specifier applies to a pointer to a `long long int` argument; or an optional `l` specifying that the following `e` . . .

Change the same paragraph, same list item, last sentence:

> If an `h`, `l`, `ll`, or `L` appears with any other conversion specifier, the behavior is undefined.

7.17.2.2 The `fwscanf` function

In paragraph 4, change the beginning of the third list item to:

> An optional `h`, `l` (ell), `ll` (ell-ell) or `L` indicating the size of the receiving object.

In the same list item, change the end of the third sentence:

> . . . , or by `l` if it is a pointer to `long int`, or by `ll` if it is a pointer to `long long int`.

Similarly, change the end of the fourth sentence:

> ..., or by **`l`** if it is a pointer to **`unsigned long int`**, or by **`ll`** if it is a pointer to **`unsigned long long int`**.

Change the last sentence of the list item to:

> If an **`h`**, **`l`**, **`ll`**, or **`L`** appears with any other conversion specifier, the behavior is undefined.

Add the following two sentences to paragraph 15:

> No argument is converted, but one is consumed. If the conversion specification with this conversion specifier is not one of **`%n`**, **`%ln`**, **`%lln`**, or **`%hn`**, the behavior is undefined.

Note to redactors and reviewers: TC1, defect report 14, added these two sentences to **`fscanf`**, but Amendment 1 failed to include them in the definition of **`fwscanf`**. This is obviously unintentional, and is corrected here due to the need to add **`%lln`** to the last sentence once it is added.

7.17.4.1 Wide-string numeric conversion functions

Add the following two new subclauses:

7.17.4.1.4 The `wcstoll` function

Synopsis

```
#include <wchar.h>
long long int wcstoll(const wchar_t *nptr, wchar_t **endptr,
      int base);
```

Description

The **`wcstoll`** function is similar to the **`wcstol`** function, except that it converts the initial portion of the wide string pointed to by **`nptr`** to **`long long int`** representation.

Returns

The **`wcstoll`** function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, **`LLONG_MAX`** or **`LLONG_MIN`** is returned (according to the sign of the value), and the value of the macro **`ERANGE`** is stored in **`errno`**.

7.17.4.1.5 The `wcstoull` function

```
#include <wchar.h>
unsigned long long int wcstoull(const wchar_t *nptr,
       wchar_t **endptr, int base);
```

Description

The **`wcstoull`** function is similar to the **`wcstoul`** function, except that it converts the initial portion of the wide string pointed to by **`nptr`** to **`unsigned long long int`** representation.

Returns

The `wcstoull` function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `ULLONG_MAX` is returned, and the value of the macro `ERANGE` is stored in `errno`.

is stored in `errno`.