Document Number: WG14 N499

C9X Revision Proposal

Title: Incorporate the `long long` integral data type into C9X

Author: Roland Hartinger

Author Affiliation: Siemens Nixdorf Informationssysteme AG

Postal Address: Otto-Hahn Ring 6, 81730 Munich, Germany

E-mail Address: Roland.Hartinger@mch.sni.de

Telephone Number: +49 . 89 . 636 44081

Fax Number: +49 . 89 . 636 44716

Sponsor: DIN

Date: 1995-22-12

Proposal category:
___ Editorial change/non-normative contribution
___ Correction
_x_ New feature
___ Addition to obsolescent feature list
___ Addition to Future Directions
___ Other (please specify) _____

Area of Standard Affected:
_x_ Environment
_x_ Language
___ Preprocessor
_x_ Library
  _x_ Macro/typedef/tag name
  _x_ Function
  _x_ Header
___ Other (please specify) _____

Prior Art: Numerous modern C compilers

Target Audience: System and application programmers

Related Documents (if any): MIPS ABI Black Book 2.0

Proposal Attached: _x_ Yes __ No, but what's your interest?

Abstract:

This document proposes a new type `long long`, an integral type with a guaranteed minimum precision of 64 bits. This follows widespread practice of compiler vendors, who, faced with their customers' requests for large integer types, preferred to extend the language rather than to change the size of an existing type.

Incorporate the `long long` integral data type into C9X

Roland Hartinger

Siemens Nixdorf Informationssysteme

ABSTRACT

This document proposes a new type **`long long`**, an integral type with a guaranteed minimum precision of 64 bits. This follows widespread practice of compiler vendors, who, faced with their customers' requests for large integer types, preferred to extend the language rather than to change the size of an existing type.

I. Introduction

Large data bases, large file systems, and object request broker systems need large integers. C9X should respond to this need by providing an integer type with a guaranteed precision of at least 64 bits.

At first glance, it might seem that this can be easily accomplished by requiring **`long`** to have at least 64 bits of precision. However, a C9X Standard that did this would force vendors to break their customers' code; not just unportable code that follows the widespread assumption of 32-bit **`long`** integers, but also any code that uses precompiled libraries and binary format data. Such a standard would meet considerable resistance from vendors and users.

Vendors can freely choose the size of the basic data types when they introduce a completely new system (on which all of the software will be new and will share the same data sizes), but changing the size or representation of a basic data type on an existing system has far-reaching consequences.

Consider what happens when a customer modifies and recompiles a program on an implementation which has just changed the size of a long from 32 bits to 64 bits. Raw binary data files break. Perhaps more importantly, calls to libraries compiled with the old implementation break (the library source file says that the argument to a particular function is a **`long`**, which was 32 bits in the old implementation, so the function in the library object file expects to be passed a 32 bit type; the library header file, included by the application, also says that the argument to the same function is a **`long`**, which is now 64 bits, so the application passes a 64 bit type).

The consequence is that the customer cannot modify an application without recompiling (if they have source) or obtaining recompiled versions of (if they do not) all of the libraries used by that application. Everything has to change at once; there is no gentle transition. For this reason, many system vendors chose to avoid changing the sizes of the basic integer types and instead established **`long long`** as a de facto standard.

C9X should follow the vendors' practice and introduce **`long long`** as a new basic integer type with a guaranteed minimal precision of 64 bits.

II. Compatibility aspects

1. ISO C++

If this proposal will be accepted by WG14 for C9X, it should also become part of the forthcoming C++ Standard. Such an extension has already been proposed to the X3J16/WG21 C++ Committee (Doc. No.: 95-0115/NO715, June 13, 1995).

2. Application Binary Interface

Proposals of 64 bit C API's for different processor architectures have already been circulated by some ABI committees. I have examined the MIPS ABI definition, the only one to which I have access. This definition requires features that are close to the ones proposed here, except that this proposal uses "ll" in place of "64" in its identifiers and leaves the exact precision of **`long long`** to the implementation.

3. External identifier length

Some of the new **`long long`** library function names are not unique in their first six or even seven characters (e.g. **`strtoull`**, **`strtoll`**). Implementations that support no more than seven significant characters in external identifiers will have to internally map these functions' names to shorter names (e.g. **`_s2ull`**, **`_s2ll`**).

III. Implementation aspects

This extension does not need any new keywords. It fits smoothly into the ISO C language and its already well-defined integral data types (**`char`**, **`short`**, **`int`**, **`long`**).

There are numerous C implementations today which already support the **`long long`** extension. These compilers have proven its implementability in practice; the extension does not burden the standardization process and the implementors with new risks in time and with undetermined costs.

If the underlying processor does not support integrals of 64 bits, **`long long`** and its operations can be emulated by software based on the existing integral types.

IV. Learning and teaching aspects

Since this is a minimal extension to the already well-defined integral data types, constants, and library functions, it also requires minimal effort to teach and learn.

V. 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 Standrad. Subclause numbers refer to the fourth 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 <u>five</u> *signed integer types*, designated as **`signed char`**, **`short int`**, **`int`**, **`long int`**, and **`<u>long long int</u>`**.

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`

Change the Semantics part, beginning with its fourth sentence, as follows:

> 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`**;

Append to the same sentence:

> 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`**.

In C89, unsuffixed decimal constants in the range from **`LONG_MAX+1`** to **`ULONG_MAX`** are assigned **`unsinged long`** type. The introduction of **`long long`** requires a design choice between **`unsigned long`** and **`long long`** type for these constants. Although selecting **`long long`** would have been more consistent with the "value preserving" style of the C Standard, it was considered more important to protect existing code that relies on the unsignedness or the size of a constant on a particular implementation.

6.2.1.5 Usual arithmetic conversions

The promotion rules for **`long long`** are analogous to those for **`long`**. They are inserted at the top of the indented section ending paragraph 1:

> 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 **`unsinged 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`**, ...

6.3 Expressions

For unsigned operands, the `~` and `<<` operators are defined in terms of arithmetic modulo an unsigned integral type's largest value plus one. These descriptions are extended to include the **`unsigned long long int`** case.

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 **`ULLONG_MAX`**, **`ULONG_MAX`**, and **`UINT_MAX`** are defined in the header **`<limits.h>`**.)

6.5.2 Type specifiers

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`**

7. Library functions

7.9.6 Formatted input/output functions

The **`fprintf`** and **`fscanf`** functions are extended to interpret **`ll`** to specify integers as **`long long`** wherever **`l`** specifies them as **`long`**. The sequence **`ll`** was preferred to the alternative, **`L`**, because it is both what programmers expect and dominant in existing practice.

7.9.6.1 The `fprintf` function

Add the following text to the fourth item near the end of paragraph 4:

> 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;

and:

> an optional **`ll`** (ell-ell) specifying that a following **`n`** conversion specifier applies to a pointer to a **`long long int`** argument;

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.

7.9.6.2 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 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.

7.9.9 File positioning functions

Since the more abstract **fgetpos** and **fsetpos** functions do not offer the same degree of control as their ancestors **fseek** and **ftell**, versions of the latter with **long long** offsets would be genuinely helpful when handling large file systems. The following two functions could be added:

7.9.9.6 The `fllseek` function

Synopsis

```
#include <stdio.h>
int fllseek(FILE *stream, long long int offset, int whence);
```

Description

The **fllseek** function is similar to the **fseek** function, except that it accepts a **long long int** offset as its second argument.

7.9.9.7 The `flltell` function

Synopsis

```
#include <stdio.h>
long long int flltell(FILE * stream);
```

Description

The **flltell** function is similar to the **ftell** function, except that it returns a **long long int** value as a large file position.

7.10 General utilities `<stdlib.h>`

Change the first two paragraph of 7.10 to introduce the type **lldiv_t** alongside **ldiv_t** and **div_t**.

> The header **<stdlib.h>** declares five types and several functions of general utility, and defines several macros.[138]
>
> 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.

7.10.1 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 functions:

7.10.1.7 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.

7.10.1.8 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**.

7.10.1.9 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**.

7.10.6 Integer arithmetic functions

7.10.6.5 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**.

7.10.6.6 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**.

7.16.4.1 Wide-string numeric conversion functions

7.16.4.1.4 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 its returned value has type **long long int**.

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**.

7.16.4.1.5 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 its returned value has type **unsigned long long int**.

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**.