(1)

From: Frank Farance
Date: 1994-06-09
Document Number: WG14/N355 X3J11/94-040
Subject: Extended Integer Range and Portable Data Structures

This proposal serves two needs:

(1) Providing known precision to applications without constraint of implementation-defined types (int, long, short, char).

(2) Providing known data structures that may be ported to different machine architecture (e.g., via FTP or NFS). This issue concerns bit/byte ordering and bit/byte alignment.

As an example, it is very difficult to write an application that portably interacts with TCP headers.

The solution to problems like this is to break apart the TCP header into known byte ordering. Then, the application must maintain a list of fields, their sizes, and their types. This is the work the compiler should be doing for manipulating data structures.

Additionally, when the application attempts to interact with the field types, 16, 32, 64-bit precision must be emulated. This is another justification for extended integer ranges.

The following types have been added:

unsigned long: N
long: N

The "long" versions produce the "fastest" type of at least N bits precision.

unsigned short: N
short: N

The "short" versions produce the "smallest" type of at least N bits precision.

unsigned int: N
int: N

The "int" versions produce arithmetic of exactly N bits. However, the "int" type may be stored in type wider than N bits. Since all types consume an exact number of bytes, identifiers declared with these extended integer ranges may have the address-of "&" operator applied to them.

The storage representation qualifiers specified the bit ordering:

lsb ← least significant bit 0
msb ← most significant bit 0

(4)

byte ordering:

bigend ← big endian
littleend ← little endian

bit alignment:

bit align: 0 adjacent bits
bit align: N round to next N bits

byte alignment:

byte align: 0 adjacent byte
byte align: N ~~addr~~round to next N bytes

Wording Changes:

6.2.1.5 Usual Arithmetic Conversions

Add:

If either operand is an extended integer range, both operands are converted to an extended integer range type of precision equal to or greater than ~~either~~ ~~one~~ both operands.

6.5.1A Storage Representation Specifier

Syntax:

~~storage~~

storage-representation-specifier:
msb
lsb
bigend
littleend

Semantics:

The storage representation specifiers allow portable representation of data. msb (most significant bit 0), and lsb (least significant bit 0) specify bit ordering of representation. bigend (big endian) and littleend (little endian) specify byte ordering of representation.

6.5.2 Type Specifiers

Syntax

~~int:~~

signed int: const-expr
unsigned int: const-expr
int: const-expr
signed long: const-expr
unsigned long: const-expr
long: const-expr
signed short: const-expr
unsigned short: const-expr

Constraints:

~~(c)~~

const-expr must be non-negative.

Semantics:

Add:

~~Unlike structure bit fields~~

... int:N has exactly N bits precision. Unlike structure bit fields, ... int:N ~~may~~ may occupy storage of more than N bits.

... long:N represents the implementation-defined type that has at least N bits precision and is regarded as the "fastest" type.

... short:N represents the implementation-defined type that has at least N bits precision and is regarded as the "smallest" type.

(8)

6.5.2.1 Structure and union specifiers

Add: Syntax

bitalign: const-expr
bytealign: const-expr

Add: Semantics

bitalign: N forces the bit alignment to round up to the next N bits.
bitalign: 0 forces the placement of the next bit field to be adjacent to the previous bit field

bytealign: N forces the byte alignment to round up to the next N bytes.
bytealign: 0 forces the placement of the next member adjacent to the previous member