

# Endian solution for C

1 November 2011, by Berwyn 2 comments

Are numbers stored big-end or little-end first? C programmers have to byte-swap manually to deal with this. Portable code becomes icky. The full ickiness is illustrated in Intel's excellent Endianness White Paper.

But in C there is no reason the compiler can't do the hard work and make programs both portable and pretty. Here we present quick hack, and also a solution requiring a minor change to the C compiler. First the quick hack.

## Quick Hack

Suppose we're dealing with a USB protocol setup packet:

```
struct setup {
  char  bmRequestType,  bRequest;
  short  wValue,  wIndex,  wLength;
};
```

Since USB is little-endian, the short integers will work on a an X86 machine, but if you have the job of porting to a big-endian ARM, you'll need to byte-swap each of these values every time they're accessed. This could be a lot of code rework.

One quick-n-dirty way to accomplish this is to simply store the USB packet in reverse order as it comes in your door (or write a reversal function). Then define your struct in reverse order:

```
struct setup {
  short  wLength,  wIndex,  wValue;
  char  bRequest,  bmRequestType;
};
```

Note that this will only work if you don't have strings in your struct, as C's string library functions don't expect strings to be in reverse order!

## A Real Solution

The following solution has the C compiler dealing with the whole endian issue for you – making your program totally portable with zero effort. It would require the addition of a single type modifier to the C compiler (C compilers already have various type modifiers). To solve endian portability, this addition would be well worth it.

This concept is similar to the 'signed' and 'unsigned' access type modifiers or the GNU C compiler's **packed** attribute which helps to access a protocol's data by letting you prevent padding between structure elements.

Our example above would become:

```
struct setup {
  char  bmRequestType,  bRequest;
  little_endian  short  wValue, wIndex, wLength;
};
```

Whenever x or y is accessed, the bytes are swapped by the compiler. You can even cast a standard long pointer to a little_endian long to force a compiler byte-swap upon access.

Internally, the compiler would probably implement just a single byte-swap type modifier which it would apply to all non-native accesses. But for portability and clarity, this should be spelled out as little_ or big_endian in the source.

It should also be noted that this same solution solves the endian problem for bit fields and bit masks (White Paper p12).

We are not C compiler gurus, so I'm not going to risk adding this change to my compiler. But I put it "out there" for comment, and hopeful uptake. I'm guessing this should go into GCC first, and then others will gradually follow suit.

## 2 comments (oldest first)

**Rodrigo** 23 Oct 2012, 11:48 link

Why don't you just use htonl() and friends ? I see it's also mentioned in the paper, but I don't understand why it isn't enough.

**Berwyn** 26 Oct 2012, 07:23 link

With htonl() the programmer has to think about network order throughout his code at every access of the data, not just at declaration time. This results in two problems: 1. Often it means the programmer will decide not to make portable code, which can be a pain later. 2. It can can create unintentional portability bugs. The programmer may think his code is portable but he's forgotten one use of htonl so that it only works on his local machine and is not portable. He misses the bug.

It also make the code noisier and thus less readable, which is never a good thing.

## Add a comment

*We reserve the right to edit or remove your comment if it's spammy, offensive, unintelligible, or if you don't give a valid email address. :-)*

**Your name**

**Email address (must be valid)**

We'll never share your email address.

**Website (optional)**

**Comment** *see Markdown formatting help*

**Add your comment**

brush technology . . . the art of good design **phone** +64 3 359 2101 **email** i...@brush.co.nz