Numeric Representations

Integer Type Sizes

We are used to thinking of a byte as 8 bits (which isn't strictly true, but is *almost always* the case), but larger sizes become more ambiguous.

It used to be the case (when 32-bit processors were dominant) that an `int` in C would be 4 bytes (32 bits), a `short int` would be 2 bytes, and a `long int` would be 8 bytes. All of these are signed quantities. `unsigned int` is the corresponding non-negative 4-byte integer value.

With most processors now being 64-bit, these have shifted somewhat. Now an `int` might be 8 bytes, though `short` and `long` may or may not be twice as long. In many programs, we don't really care, but when we're encoding numbers, this becomes very important.

The header file `stdint.h` contains the following types, which you should use when you want to ensure the size of the value in bytes:

Type	Size (bytes)	Signed/Unsigned
`int8_t`	1	signed
`int16_t`	2	signed
`int32_t`	4	signed
`int64_t`	8	signed
`uint8_t`	1	unsigned
`uint16_t`	2	unsigned
`uint32_t`	4	unsigned
`uint64_t`	8	unsigned

Byte Encoding

Numbers have to be stored in memory on a host. They also have to be saved in files and sent over the network. This seems simple, but how a number is stored is more complicated than you might expect.

While a single-byte integer value is easy ("10" is "`0A`" in hex), once you have more than one byte, you have to consider the specific *architecture*. There are two main architectures commonly used: *big endian* (BE) and *little endian* (LE). In big endian encoding, the most significant byte of the number comes first in memory. In little endian encoding, the least significant byte come first.

Some examples might help:

Number	Size (bytes)	BE	LE
12	2	`00 0C`	`0C 00`
3072	2	`0C 00`	`00 0C`
4660	2	`12 34`	`34 12`
13330	2	`34 12`	`12 34`
12	4	`00 00 00 0C`	`0C 00 00 00`
201326592	4	`0C 00 00 00`	`00 00 00 0C`

Host and Network Byte Order

The host's architecture specifies the *host byte order*, but when exchanging values over the network, we can't have architecture-dependent ambiguity. Consequently, the networking community decided on big endian as the standard *network byte order*.

Because of this, if we receive a 4-byte integer value `0000000C`, we can safely assume these bytes represent the number 12, not 201326592, regardless of how our host interprets this sequence of bytes.

Converting Between Encodings

The C standard library has a number of functions to handle conversions between BE and LE encoding. Other languages have their own mechanisms, which you can look up if you need them. Here is a summary (header files might vary from system to system):

Function	Size (bytes)	Input Encoding	Output Encoding	Header
`htons`	2	host	network	`arpa/inet.h`
`ntohs`	2	network	host	`arpa/inet.h`
`htonl`	4	host	network	`arpa/inet.h`
`ntohl`	4	network	host	`arpa/inet.h`
`htobe16`	2	host	big endian	`endian.h`
`htole16`	2	host	little endian	`endian.h`
`be16toh`	2	big endian	host	`endian.h`
`le16toh`	2	little endian	host	`endian.h`
`htobe32`	4	host	big endian	`endian.h`
`htole32`	4	host	little endian	`endian.h`
`be32toh`	4	big endian	host	`endian.h`
`le32toh`	4	little endian	host	`endian.h`
`htobe64`	8	host	big endian	`endian.h`
`htole64`	8	host	little endian	`endian.h`
`be64toh`	8	big endian	host	`endian.h`
`le64toh`	8	little endian	host	`endian.h`