Serialization and Bit Operations CS 2022: Introduction to C

Instructor: Hussam Abu-Libdeh

Cornell University (based on slides by Renato Paes Leme)

Fall 2011, Lecture 10

Serialization

Sending data between programs

- Disk
- Network
- Pipes
- Between programs on multiple hosts
 - Different endianness
 - Different architectures

Binary vs. Text

Binary...

- Compact
- Easy to encode/decode
- Faster
- e.g. IP, TCP, AIM, ...

Text...

- Easily debugged
- (Can be) self-documenting
- Arch/Endian independent
- e.g. HTTP, SMTP, MSN

Ok, but how?

What serialization solution to use?

- tpl library
- c11n library
- Google protocol buffers
- Customized solution

Which standard to use?

- ► XML, XDR, protocol buffer, ...
- Network protocol standards

Handling Endianness

Decimal: 3735928559 Binary: 11011101010110110111011101111 Hex: Oxdeadbeef Big Endian: Oxde Oxad Oxbe Oxef Little Endian: Oxef Oxbe Oxad Oxde

Always in big-endian form when loaded into the CPU

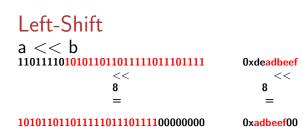
000000000000000010111110000000000

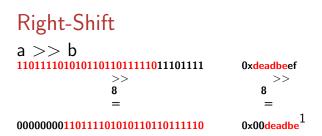
0xdeadbeef & 0x0000FF00 =

0x0000be00



Serialization and Bit Operations CS 2022, Fall 2011, Lecture 10





¹for unsigned ints only. For signed ints, the instead of zero-padding, the top-most bit is repeated

Serialization and Bit Operations CS 2022, Fall 2011, Lecture 10

Compliment (flips bits)

~a ~110111101011011011011111011111 ~0xdeadbeef = = = 00100001010100100100000100010000 0x21524110

2's compliment representation for negative numbers: -x = \sim x + 1

Exercise 1

int htonl(int x) {
 int b1, b2, b3, b4, y;

return y;

Serialization

- Use structures for data-types
- Copy data in one-go memcpy(dst, src, numbytes)
- Use standard (big) endianness for multi-byte variables
- NEVER serialize pointer values. Why?

Tricks with bits

- ► How to iterate over all sets $S \subseteq \{0, 1, 2, \dots, k-1\}$?
- There are 2^k such sets. I just need one for to do that.
- ► Think of a number 0 ≤ 0 < 2^k in binary. It represents a subset of S.
- Given a subset *S*, let $a_i = \begin{cases} 1, i \in S \\ 0, i \notin S \end{cases}$, then we represent *S* by $\sum_{i=0}^{k-1} a_i 2^i$.

Tricks with bits

How to iterate over all sets $S \subseteq \{1, \ldots, n\}$?

Given two sets A and B represented as binary strings:

- Union:
 - A | B
- Intersection:
 - A & B
- Single element set {1}:

1 << i

• Testing $i \in A$:

A & (1 << i) != 0

Given two sets A and B represented as binary strings:

Adding element i to A:

A = A | (1 << i)

• Removing element *i* from *A*:

A = A & ~(1 << i)

▶ Toggle element *i* in *A*:

$$A = A ^{(1 << i)}$$

Given S a string representation of a set, how to iterate over all its subsets $T \subseteq S$:

More complicated exercise: how to iterate over all subsets of $\{1, \ldots, n\}$ of size k.

```
int s = (1 << k) - 1;
while (!(s & 1 << N))
{
    // do stuff with s
    int lo = s & ~(s - 1); // lowest one bit
    int lz = (s + lo) & ~s; // lowest zero bit above lo
    s |= lz; // add lz to the set
    s &= ~(lz - 1); // reset bits below lz
    s |= (lz / lo / 2) - 1; // put back right number of bits at end
}</pre>
```

Source of bit tricks

A bit of fun, fun with bits: http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=bitManipulation

Other iteration exercises

- 1. Write a code that iterates over $\{0, \ldots, n-1\}^k$, i.e., all *k*-uples (t_1, \ldots, t_k) where $0 \le t_i < n$.
- 2. Write a code that iterates over $\{0, \ldots, n-1\}^k$, i.e., all k-uples (t_1, \ldots, t_k) where $0 \le t_i < n$ and $t_1 \ge t_2 \ge \ldots \ge t_k$.
- Write a code that iterates over all the permutations of {1,..., n} and writes them on the screen.