CSE 311 Foundations of Computing I

Lecture 10
Divisibility and Modular Arithmetic
Autumn 2011

Announcements

- Reading assignments
  - Today:
    - 4.1-4.2 7th Edition
    - 3.4, 3.6 up to p. 227 6th Edition
    - 2.4, 2.5 up to p. 177 5th Edition
- Homework 4
  - Coming soon . . .

Highlights from last lecture

- Set theory and ties to logic
- Lots of terminology
  - Complement, Universe of Discourse, Cartesian Product, Cardinality, Power Set, Empty Set, N, Z, Z^+, Q, R, Subset, Proper Subset, Venn Diagram, Set Difference, Symmetric Difference, De Morgan's Laws, Distributive Laws
- Bit vector representation of characteristic functions
  - Bitwise operations for Set operations

Unix/Linux file permissions

- **`ls –l`**

```
drwxr-xr-x ... Documents/
-rw-r--r-- ... file1
```

- Permissions maintained as bit vectors
  - Letter means bit is 1 – means bit is 0.
- How is **`chmod og+r`** implemented?

A simple identity

- If x and y are bits: $(x \oplus y) \oplus y = ?$
- What if x and y are bit-vectors?

Private Key Cryptography

- Alice wants to be able to communicate message secretly to Bob so that eavesdropper Eve who hears their conversation, cannot tell what Alice's message is
- Alice and Bob can get together and privately share a secret key K ahead of time.

One-time pad

- Alice and Bob privately share random n-bit vector K
  - Eve does not know K
- Later, Alice has n-bit message m to send to Bob
  - Alice computes $C = m \oplus K$
  - Alice sends C to Bob
  - Bob computes $m = C \oplus K$ which is $(m \oplus K) \oplus K$
- Eve cannot figure out m from C unless she can guess K

Russell's Paradox

$$S = \{ x \mid x \notin x \}$$

Number Theory (and applications to computing)

- Branch of Mathematics with direct relevance to computing
- Many significant applications
  - Cryptography
  - Hashing
  - Security
- Important tool set

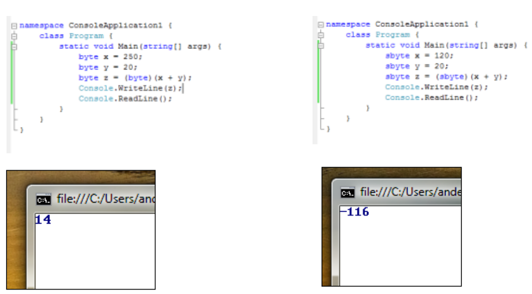
Modular Arithmetic

- Arithmetic over a finite domain
- In computing, almost all computations are over a finite domain

What are the values computed?

```
public void Test1() {
        byte x = 250;
        byte y = 20;
        byte z = (byte) (x + y);
        Console.WriteLine(z);
}
```

```
public void Test2() {
        sbyte x = 120;
        sbyte y = 20;
        sbyte z = (sbyte) (x + y);
        Console.WriteLine(z);
}
```



Arithmetic mod 7

- a $+_7$ b = (a + b) mod 7
- a $\times_7$ b = (a $\times$ b) mod 7

+	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

x	0	1	2	3	4	5	6
0							
1							
2							
3							
4							
5							
6							

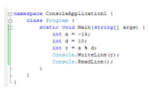
Divisibility

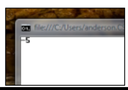
Integers a, b, with a ≠ 0, we say that a *divides* b is there is an integer k such that b = ak. The notation a | b denotes a divides b.

Division Theorem

Let *a* be an integer and *d* a positive integer. Then there are *unique* integers *q* and *r*, with $0 \le r < d$, such that $a = dq + r$.

q = *a* **div** *d*   *r* = *a* **mod** *d*





Note: r ≥ 0 even if a < 0. Not quite the same as `a%d`

Modular Arithmetic

Let a and b be integers, and m be a positive integer. We say a *is congruent to b modulo m* if m divides a – b. We use the notation a ≡ b (mod m) to indicate that a is congruent to b modulo m.

Modular arithmetic

Let a and b be integers, and let m be a positive integer. Then a ≡ b (mod m) if and only if a mod m = b mod m.

Modular arithmetic

Let m be a positive integer. If a ≡ b (mod m) and c ≡ d (mod m), then
- a + c ≡ b + d (mod m) and
- ac ≡ bd (mod m)

Example

Let n be an integer, prove that $n^2 \equiv 0 \pmod 4$ or $n^2 \equiv 1 \pmod 4$