CMPE-013/L

Introduction to “C” Programming

Bryant Wenborg Mairs

Spring 2014
Hashing
Hashing

- Mapping data of arbitrary size into a fixed-size hash value
- Utilizes a hash function
- Effectively mapping values from a higher-dimensional space into a lower one
- Produces aliasing
Hashing

Example

Inputs

keys

hash function

hashes

"John Smith"

"Lisa Smith"

"Sam Doe"

"Sandra Dee"

00

01

02

03

04

05

15
Hashing

Uses

• CPU caches

• Datatypes: hashmap/dictionary

• Data verification: fingerprinting

• Data compression: vector quantization
Hashing

8-bit XOR ()

Example:

16-bit data

\[ \oplus \rightarrow \oplus \rightarrow \text{XOR()} \]

16-bit data

\[ \text{struct } \equiv \text{uint16} \equiv \text{uint16} \rightarrow \text{hash()} \rightarrow \text{evid} \]
Checksums
Checksums

Definition

• A small piece of data computed from an original source of data for the purposes of verifying it
• Can utilize hashing
• Relies on a checksum algorithm
Checksums

Uses

• Verify data transmit over radio
  – Such as in a telemetry stream for a robot
• Verify the integrity of a data burned to a CD
• Verify correctness of a file downloaded off the internet

.zip archive: apache-ant-1.9.4-bin.zip [PGP] [SHA1] [SHA512] [MD5]
Checksums
Checksum functions

• SHA512 ≤ SHA-2 cryptographically secure
  – 512-bits

• MD5
  – 128-bits

• XOR ≤ used in protocol
  – Usually wordsize to simplify computation, between 8- and 64-bits
Checksums
Using checksums

• When used in message transmission, transmit both the data and the checksum
Checksums
Using checksums

• On message reception, recalculate the checksum and verify that it matches the one transmit.
Checksums
XOR Checksum in C

Syntax

```c
uint8_t CalcStringChecksum(const char *data);
```

Example

```c
char *str = "Mary had a little lamb.";
uint8_t strChecksum = CalculateStringChecksum(str);
printf("XOR(%s) = %02X\n", str, strChecksum);
```
Random number generation
Random number generation

Usage

• Pretty much all games
  – Described with "randomness" and "variation"
• Security and cryptography
• Problem solving algorithms
• Music/video playback
• Recommendation systems
• User interfaces
Random number generation

Categories

• "True" random
  – Result of noisy physical phenomena
  – No initial input (besides, possibly, power)
  – No repeatable sequence
  – Not in the C standard

• Pseudo-random
  – Result of algorithm
  – Relies on initial (seed) value
  – Produces cycles of numbers
  – In the C standard
Random number generation

Functions

Syntax

```c
void srand(unsigned int seed);
```

- `seed` is the initial value to iterate on
  - Remembered until next call to `srand()`
Random number generation

Functions

Syntax

```c
int rand(void);
```

- Returns pseudo-random number based on seed
  - Values between \texttt{INT\_MIN} and \texttt{INT\_MAX}
  - See set by \texttt{srand()} otherwise defaults to 1
- All \texttt{rand()} calls with the same seed produce the same sequence.
Random number generation

Real-world example

```c
void main()
{
    srand(67);
    int truth = rand(), guess;
    do {
        printf("Guess the number:");
        scanf("%f", &guess);
        if (guess == truth) {
            printf("You win!\nTry again.");
            truth = rand();
        }
    } while (1);
}
```
Random number generation

Initial seed

• But how do we choose a good initial seed?

• Hardcode it
  – The PS3 problem

• Fake it
  – Use compile-time information like __DATE__
    and __TIME__
  – Use data that changes
    • Current date/time
    • User input
    • Physical sensors
// The first part of our seed is a hash of the compilation
// time string.
char seed1[] = __TIME__; // initial seed from compilation
int seed1Len = strlen(seed1);
int firstHalf = seed1Len / 2;
uint16_t seed2 = 0;
int i;
for (i = 0; i < seed1Len; ++i) {
    seed2 ^= seed1[i] << ((i < firstHalf) ? 0 : 8);
}

// Now we hash in the time since first user input (which, as // a 32-bit number, is split and each half is hashed in // separately).
srand(seed2 ^ (counter >> 16) ^ counter); // combine wr user input
Random number generation

Hardware crypto on the PIC32MZ

- The PIC32MZ series has hardware RNG
Encryption
Encryption

• Encoding data such that only agents with a key can access it
• Used everywhere
  – Especially now with the NSA's shenanigans
• Relies on computational complexity and secret knowledge

\[ \text{By Dan Brown, } \text{"Digital Fortress"} \]
Encryption

Types

• Multiple types of encryption:
  – Public key – Separate keys for encryption and decryption
  – Private/Symmetric key – Same key used for encryption and decryption
Encryption

Public key

• Separate keys for encryption and decryption
• Encryption key is public
  – Anyone can encode
• Decryption key is private
  – Only authorized parties can decode
Encryption

Public key

Bob

Hello Alice! → Encrypt → Alice's public key

6EB69570 08E03CE4

Alice

Hello Alice! → Decrypt → Alice's private key

CMPE-013/L: “C” Programming

Gabriel Hugh Elkaim – Winter 2014
Encryption

Symmetric key

• Single key for encryption and decryption
• Key needs to be kept private by all parties
Encryption

Encryption function

- The operation for encrypting from a key must be known for encryption and decrypting
- Simplest bidirectional function is \( \text{xor}(\)\)

\[
\begin{align*}
\text{Data} & \quad n+1 = n \times \text{mod}(b) + c \\
\text{Key} & \quad \text{0b11001010} \\
\text{xor()} & \quad \rightarrow \quad \text{0b01100000} \\
\text{0b10101010} & \quad \rightarrow \quad \text{encrypted data}
\end{align*}
\]
Encryption
Symmetric key example

• If Alice and Bob want to communicate, both need to agree on the private key.
Encryption
Symmetric key example

• If Alice and Bob want to communicate, both need to agree on the private key.
Encryption
Symmetric key example

- If Alice and Bob want to communicate, both need to agree on the private key.
Encryption
Real-world example

• Problem: Two agents need to determine which goes first. Don't allow cheating
• Emulate flipping a coin
  – Agents each guess a number, depending on those numbers either the higher or lowest number wins
• Problem is time:
  – In real world systems, no event occurs simultaneously
  – If an agent sends their guess first, the other agent can cheat by choosing their guess appropriately
Encryption
Real-world example

• Solution: Split the guessing into 2 stages
  – Send an encrypted guess
  – After receiving the other agent's guess, send your decryption key.

• New problem:
  – If agent receives other agent's guess & key, they could cheat by generating a new guess and key that still has the same encrypted value (which they've already sent)
Encryption
Symmetric key example

Alice

Decrypted data

Regenerate key & guess

time

Encrypted data & id

Encryption key

Bob

Encrypted data & id

Encryption key

Regenerate key & guess

encryption

hash

Alice wins!
Encryption
Real-world example

• Solution: Also send a pseudo-unique identifier of the key/guess pair

• New problem:
  – If agent receives other agent's guess & key, they could cheat by generating a new guess and key that still has the same encrypted value (which they've already sent)
Encryption

Symmetric key example

```
Alice

Decrypted data
Regenerate key & guess

time

Bob

Encrypted data & id
Encryption key & guess

Bob detects cheating!

Bob verifies Alice's data

need to match
```
Communications
Communications

• Communications can almost never be assumed to be simultaneous
  – Due to real-time constraints
  – Technical limitations

• Systems require synchronization
  – Handled with state machines
Communications

Between two agents

Alice

Bob

time
Communications
With a protocol

- Bob needs to ACK after receiving an IMP message
Communications

With a protocol

• But what if Bob is busy? Maybe receiving more data from Alice?
Communications

With a protocol

• An FSM can be used for remembering than an ACK needs to be sent
Communications

With a protocol

- An FSM can be used for remembering than an ACK needs to be sent

Alice

- IMP
- WAITING

Bob

- REC_IMP
• An FSM can be used for remembering than an ACK needs to be sent
Communications

With a protocol

• An FSM can be used for remembering than an ACK needs to be sent
Communications

With a protocol

- An FSM can be used for remembering than an ACK needs to be sent

Alice

Bob

waiting

waiting

imp

rec_imp

sent_ack

ack

time