CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne

Spring 2016
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

<table>
<thead>
<tr>
<th>keys</th>
<th>hash function</th>
<th>hashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td></td>
<td>00</td>
</tr>
<tr>
<td>Lisa Smith</td>
<td></td>
<td>01</td>
</tr>
<tr>
<td>Sam Doe</td>
<td></td>
<td>02</td>
</tr>
<tr>
<td>Sandra Dee</td>
<td></td>
<td>03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
Hashing

Uses

- CPU caches
- Datatypes: hashmap/dictionary
- Data verification: fingerprinting
- Data compression: vector quantization
Hashing
8-bit XOR

16-bit data → XOR() → 8-bit hash
16-bit data
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**
  - 512-bits

- **MD5**
  - 128-bits

- **XOR**
  - Usually wordsize to simplify computation, between 8- and 64-bits
100 Hz: \( BE = \text{Buttons \_ check}(\text{Event}) \)

5 Hz: \( \text{FR} \uparrow \uparrow \);

Slow it down

2 Hz: 2 Hz \( \text{event} = \text{TRUE} \);

\( \text{time} \uparrow \uparrow \)

\( \text{if}(\text{time} < 0/0.2) \)
Mode = "Toast"

2 H2 triggered

strcmp

Mode = TOAST

time: 30 < 60
(Update display)

switch

state machine

(update needed) = TRUE

if (update needed)
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);
```
Parity bit
1-bit checksum
number of ones  odd or even

011 011 011
1 2 3 4 5

ISBN
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

(radiation)
(lightning)
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 `INT_MIN` and `INT_MAX`
  - See set by `srand()` otherwise defaults to 1
- All `rand()` calls with the same seed produce the same sequence.
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__
    - Use data that changes
      - Current date/time
      - User input
      - Physical sensors

A/D to air
// The first part of our seed is a hash of the compilation
// time string.
char seed1[] = __TIME__;
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);
Random number generation

Hardware crypto on the PIC32MZ

- The PIC32MZ series has hardware RNG
Press your luck  1943

Larsen

110,000

3-4,000
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
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
6EB69570 08E03CE4

Alice's public key

Alice

Hello Alice!
Decrypt

Alice's private key
Encryption
Symmetric key

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

Bob

Alice

Hello Alice!
Encrypt
6EB69570 08E03CE4

Hello Alice!
Decrypt
Secret key
non-electronic
Encryption

Encryption function

- The operation for encrypting from a key must be known for encryption **and** decrypting
- Simplest bidirectional function is `xor()`
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

- Encrypted data & id
- Decrypted data
- Regenerate key & guess
- Time

Bob

- Encryption key
- Encryption key
- Encryption data & id

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

Encrypted data & id

Encryption key & guess

Bob

Encrypted data & id

Encryption key & guess

Bob verifies Alice's data

Bob detects cheating!
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

Alice

Bob

IMP

ACK

time
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

Alice

Bob

WAITING

time
Communications

With a protocol

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

Alice  -->  Bob

| WAITING |
|        |

IMP  -->  REC_IMP

time
Communications

With a protocol

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

Alice

Bob

WAITING

IMP

REC_IMP

time
Communications

With a protocol

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

Alice

Bob

WAITING

REC_IMP

SENT_ACK

time
Communications
With a protocol

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

Alice

Bob

waiting

IMP

REC_IMP

SENT_ACK

waiting

ACK

time
File I/O
File formats
Void pointers
Function Pointers
Unions