CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
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 01</td>
</tr>
<tr>
<td>Lisa Smith</td>
<td></td>
<td>02 03</td>
</tr>
<tr>
<td>Sam Doe</td>
<td></td>
<td>04 05</td>
</tr>
<tr>
<td>Sandra Dee</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
Hashing

Uses

- CPU caches
- Datatypes: hashmap/dictionary
- Data verification: fingerprinting
- Data compression: vector quantization

```c
item["Foo"]
```
Hashing
8-bit XOR

16-bit data → XOR() → 8-bit hash
16-bit data → XOR() → 8-bit hash
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]

Handbrake
Checksums
Checksum functions

- SHA512
  - 512-bits
- MD5
  - 128-bits
- XOR
  - Usually wordsize to simplify computation, between 8- and 64-bits
- CRC32
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

Lightning AM

Mersenne Twister
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 \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

Hypervisor
acid etched
10 million LA

Maxwell James Dunne
Random number generation
Real-world example

// 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
Random number generation

Difference between random and pseudo-random (taken from Random.org)
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
  - "swimming pool"
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

.pub
.id_rsa
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

Hello Alice!

Encrypt

6EB69570 08E03CE4

Secret key

Decrypt

Hello Alice!

Bob

Alice
Bitcoins

**Encryption**

Encryption function

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

GPU 5,000

```
0b101010110 → xor() → 0b01100000
```
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

Bob

Encrypted data & id

Encryption key

Encryption key

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

Encrypted data & id

Bob

Encrypted data & id

Encryption key & guess

Decrypted data

Encryption key & guess

Regenerate key & guess

Bob verifies Alice's data

Bob detects cheating!

Maxwell James Dunne
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?

Alice → Bob

IMP

ACK

time
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
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

WAITING
CMPE-013/L

Toaster Oven Lab

Maxwell James Dunne
No floating point percentages

\[ \frac{100 \times (\frac{x}{10})}{10} \]
Integer Timing

Free running counters and precision

$30 \quad 29 \quad 28$

$2 \text{ Hz} \approx \frac{1}{2} \text{s}$

$\text{float time } x$

$x = -.5$;

60 59 58
time interval
free running timer
ISR
count += j

record count
newcount = count - \text{diff}
unsigned byte

254 ← 255
    0
    1
    2 ←

2 - 254
-252

int 22 billion

1ms 16 bit 2 min
32 bit 1.8 months
OLED Display

Formatting and Update Cycles

format everywhere

Print Screen

```
0/0 < 0/0 < 0/0 < 0/0 < 0/0 < 0/0 < 0/0 < 0/0 <
\x4, \x4, \x4, \x4, \x4
0/0/5
```
While(1)
ix (E != 0)
Print event
sprintf("\0x01\0x01\0x01\0x01\0x01\0x01\0x01", ...)

0 %

cooking time left / 60

0 / 60