CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Spring 2016
Software Engineering

Design

Build
Software Engineering

Design process

Initial Planning
Requirements
Analysis/Design
Implementation
Testing
Evaluation

Deployment

Turnin
Software Engineering

Principles

• Use **consistent styling**

• Summary:
  – Utilize whitespace
  – Good variable/function names
  – Comments that describe non-obvious code behavior
    • "How?" and "why?" are good questions to answer in comments

```
i++  // i = i + 1
```
Software Engineering

Formatting code

- Ugly code
- Beautiful code
Software Engineering

Formatting non-code

- Comments that describe non-obvious code behavior
  - "How?" and "why?" are good questions to answer in comments

```
// First, determine the length of both items' data,
// given NULL data a -1 length so that it sorts to
// the head of the list.
int len1 = -1;
if (item1->data) {
    len1 = strlen(item1->data);
}
...```
Software Engineering

Principles

- Modularity is important

- Why?
  - Supports code reuse
  - Simplifies changes
  - Allows for testing

- How?
  - Keep functions small
  - Minimize side effects
  - Information hiding/encapsulation

Leds.h
Buttons.c
3 files
Software Engineering

Principles

• Information hiding/encapsulation

• Summary:
  – Hide unimportant details from the user
  – Protects the user from breaking things
  – Separates backend from frontend

LEDS, $\text{set}(C)$
Software Engineering

Mantras

- Keep it simple, stupid
  - KISS
- Summary:
  - Don't solve problems you don't need to
  - Don't introduce unnecessary complexity
  - Prioritize for readability and modularity
  - Don't be clever and/or cute
  - Applies to code architecture and specific code constructs

For ever
$AD$ not inside atimer
Software Engineering

KISS example

```
ListItem *LinkedListGetFirst(ListItem *list)
{
    ListItem *tempPointer = NULL;
    if (list == NULL) {
        return NULL;
    }
    if (list->previousItem == NULL && list->nextItem != NULL) {
        return list;
    } else if (list->previousItem != NULL) {
        tempPointer = list;
        while (tempPointer->previousItem != NULL) {
            tempPointer = tempPointer->previousItem;
        }
    }
    return tempPointer;
}
```
Software Engineering

KISS example

```c
ListItem *LinkedListGetFirst(ListItem *list) {
    while (list && list->previousItem) {
        list = list->previousItem;
    }
    return list;
}
```
Software Engineering

Mantras

• Don't repeat yourself
  – DRY

• Summary:
  – Write code only once
  – Simplifies refactoring/incremental development
  – Avoids copy/paste errors
Software Engineering

Mantras

• You aren't gonna need it
  – YAGNI

• Summary:
  – Don't introduce features that are unnecessary
  – Don't write more code than you have to
  – Start small and build from there
Software Engineering

Principles

• Principle of Least Astonishment

• Summary:
  – Be consistent with user's expectations
  – Build on user's intuition
  – Applies to users and developers
    • so both the code and library/program functionality
  – Lowers learning curve

ADCInit
IsAllChosen
Software Engineering
Principle of Least Astonishment

• Functions/variables should have clear names
  – That should match their functionality!
  – Same for comments
• Functions should not do more than you would think
  – Minimize side effects
• Code should be grouped logically
• Functionality should follow precedence if any exists
Software Engineering

Principles

• Garbage in, garbage out

• Summary:
  – "A system's output quality usually cannot be better than the input quality"
  – So bad input results in garbage output
    • Instead of an error condition
  – Can propagate through the system
  – Can be mitigated by checking the input data
Software Engineering

Principles

• Fault tolerant design

• Summary:
  – Plan for operating failures
    • Running out of memory
    • Data being corrupted
  – Provide fallback modes
  – Important for complex software where minor errors can be common
  – Part of defensive programming

if (malloc)

FATAL_ERROR()
Error tolerant design

Summary:

- Plan for user errors
  - "Fault tolerant design" applied to the human component
- Primarily invalid user input
- Important for complex software where minor errors can be common
- Part of defensive programming
Software Engineering
Writing fault/error tolerant code

• Check return values for errors!
  – Many functions have special return values when there are errors, these should usually be checked
  – File accesses
  – scanf()
  – malloc()

• Your code should have special error values
  – LinkedList library

• Program should also return error if failure
Software Engineering

Writing fault/error tolerant code

- Errors should be exposed by libraries

**Good library**

```c
int LinkedListSwapData(ListItem *firstItem,
                        ListItem *secondItem);
int LinkedListSort(ListItem *list);
int LinkedListListPrint(ListItem *list);
```

**Bad library**

```c
void LinkedListSwapData(ListItem *firstItem,
                        ListItem *secondItem);
void LinkedListSort(ListItem *list);
void LinkedListListPrint(ListItem *list);
```
Software Engineering
Writing fault/error tolerant code

- Errors should be exposed by libraries
- And handled by the program
- Not all errors can be recovered from
  - Fatal errors

Embedded example

```c
int main(void) {
    if (!DataStoreInit()) {
        FATAL_ERROR();
    }
}
```

With OS example

```c
int main(void) {
    if (!DataStoreInit()) {
        return DATASTORE_ERROR;
    }
}
```
Software Engineering

Principles

Google

• Eating your own dogfood

• Summary:
  – When engineers use their own creations, they're generally better
  – More likely that bugs are fixed, features are added because they directly impact the developers
  – In use by all of industry
  – I do it
Software Engineering

Pitfalls

• Premature Optimization
  - "root of all evil"

• Summary:
  - Optimizing code before performance is a critical factor
  - Optimizing reduces readability & modularity
  - Optimization not required for a lot of code
    • See Amdahl's Law
  - See KISS

for(l = 0; l < 4; l++)
Software Engineering

Teamwork

• Working as a group is the most challenging engineering practice
• Requires:
  – Good communication
• That's it!
Software Engineering

Teamwork

• Pair programming
• Summary:
  – Two developers work side by side: one driving, the other navigating
  – Just like driving:
    • Driver writes code
    • Navigator plans ahead, thinks of edge cases, double-checks driver
  – Requires frequent role switching to be effective!
Meat served by an expert novice.

\[ V \]
Software Engineering

Teamwork

- Division of labor
- Summary:
  - Divide work into tasks that can be split between team members
  - Requires coordination to not step on each other's toes
  - Documentation is very important!
  - Can be useful to split testing and development between different people
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>01</td>
</tr>
<tr>
<td>Lisa Smith</td>
<td></td>
<td>02</td>
</tr>
<tr>
<td>Sam Doe</td>
<td></td>
<td>03</td>
</tr>
<tr>
<td>Sandra Dee</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
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**

```
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.
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__;
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
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

6EB6957008E03CE4

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
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

Decrypted data

Regenerate key & guess

time

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
Decrypted data
Regenerate key & guess

time

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
Communications
With a protocol

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

Alice

Bob

IMP

ACK

time

Maxwell James Dunne – Spring 2016
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

Maxwell James Dunne – Spring 2016
Communications

With a protocol

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

Alice  ↓  time  ↓  Bob

IMATION

WAITING

REC_IMP
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

IMP

REC_IMP

ACK

SENT_ACK

WAITING

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