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 02</td>
</tr>
<tr>
<td>Sam Doe</td>
<td></td>
<td>03 04</td>
</tr>
<tr>
<td>Sandra Dee</td>
<td></td>
<td>05 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: RE = Button5_checkEvent5()

5 Hz: FR++;

2 Hz: 2 Hz event = TRUE;

**Slow it down**

\[ \text{time}++; \]

\[ i < (\text{time} < 0.2) \]
Mode = "Toast"

2H2 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 01101
12 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
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

```
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?
  • Hardcore it
    • The PS3 problem
  • Fake it
    – Use compile-time information like ___DATE___
    – Use data that changes
      • Current date/time
      • User input
      • Physical sensors

button on 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 1983

Larsen

\[ \frac{110,000}{3-4,000} \]

\[ \text{WH} \]

\[ \text{Money} \]
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

Bob

Hello Alice! → Encrypt → 6EB69570 08E03CE4

Alice

Hello Alice! → Decrypt → non-electronic
Encryption

Encryption function

- The operation for encrypting from a key must be known for encryption \textbf{and} decrypting
- Simplest bidirectional function is \texttt{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

Encrypted
data & id

Encryption
key

Encryption
key

Alice wins!

Encrypted
data & id

Bob
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

encrypted data & id

Decrypted data

Regenerate key & guess

Encryption key & guess

Encryption key & guess

Bob

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
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.
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

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
File I/O
File formats
Void pointers
Function Pointers
Unions

Variadic functions

GCC

HSM
Advanced Language Concepts

- Unions
- Function pointers
- Void pointers
- Variable-length arguments
- Program arguments
Unions
Unions allow the same piece of memory to be used as different datatypes in different contexts. A single union can hold any datatype that is in its declaration.

- Unions:
  - May contain any number of members of any type
  - Are as large as their largest member
  - Initializing uses the datatype of its first member
  - Use exactly the same syntax as structures except `struct` is replaced with `union`
Unions
Creating unions

Syntax

```
union UnionName {
    type1 memberName1;
    ...
    typen memberName_n;
};
```

Example

```
union MixedBag {
    char a; 8
    int b; 32
    float c; 32
};
```
Unions

Unions and *typedef*

**Syntax**

```c
typedef union UnionTag_{optional} {
    type_{1} memberName_{1};
    ...
    type_{n} memberName_{n};
} typeName;
```

**Example**

```c
typedef union {
    char a;
    int b;
    float c;
} MixedBag;
```
Unions

Initializing unions

**Syntax**

```c
union UnionName {
    type1 memberName1;
    ...
    typen memberNamen;
} variableName = {VALUE};
```

**Example**

```c
union MixedBag {
    char a;
    int b;
    float c;
} myBag = {'a'};
```
Unions
In memory

- Memory is only allocated to accommodate the union’s largest member

Example

```
typedef union {
    char a;
    short b;
    float c;
} MixedBag;

MixedBag x;
```

Space allocated for `x` is `sizeof(float)`

Data Memory (RAM)

```
0x800 0x804 0x808 0x80C
```

X
Unions
In memory

- Memory is only allocated to accommodate the union’s largest member

```c
typedef union {
    char a;
    short b;
    float c;
} MixedBag;

MixedBag x;
```

Data Memory (RAM):
- `x.a` only occupies the lowest byte of the union.
Unions

In memory

- Memory is only allocated to accommodate the union’s largest member

Example

```c
typedef union {
    char a;
    short b;
    float c;
} MixedBag;

MixedBag x;
```

x.b only occupies the lowest two bytes of the union
Unions
In memory

- Memory is only allocated to accommodate the union’s largest member

```
typedef union {
    char a;
    short b;
    float c;
} MixedBag;

MixedBag x;
```

Example

<table>
<thead>
<tr>
<th>Data Memory (RAM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x800</td>
</tr>
<tr>
<td>0x804</td>
</tr>
<tr>
<td>0x808</td>
</tr>
<tr>
<td>0x80C</td>
</tr>
</tbody>
</table>

X. C occupies all four bytes of the union
Unions
Accessing members

Example

typedef union {
    char a;
    int b;
    float c;
} MixedBag;

MixedBag myBag = {'a'};
printf("myBag: char=%c, int=%d, float=%f",
    myBag.a, myBag.b, myBag.c);
Unions
Real-world example

Example: Binary tree for storing chars, ints, or floats

typedef union {
    char asChar;
    int asInt;
    float asFloat;
} AnyData;

typedef enum {
    CHAR,
    INT,
    FLOAT,
} DataType;

typedef struct Node {
    struct Node *leftChild;
    struct Node *rightChild;
    DataType type;
    AnyData data;
} Node;
Function pointers
Function Pointers

- Pointers may also be used to point to functions
  - Because it's just a memory address
- Provides a more flexible way to call a function, by providing a choice of which function to call
- Makes it possible to pass functions to other functions
- Not extremely common, but very useful in the right situations
Function Pointers

Declaration

- A function pointer is declared much like a function prototype:

\[
\text{int (}*\text{fp})(\text{int } x); \\
\]

- Here, we have declared a function pointer with the name \text{fp}
  - The function it points to takes one int parameter
  - The function it points to returns an int
Function Pointers

Initialization

- A function pointer is initialized by setting the pointer name equal to the function name.

If we declare the following:

```c
int (*fp)(int x); // Function pointer
int Foo(int x);   // Function prototype
```

We can initialize the function pointer like this:

```c
fp = Foo;       // fp now points to Foo
```
Function Pointers

Calling a Function via a Function Pointer

• The function pointed to by fp from the previous slide may be called like this:

   \[ y = \text{fp}(x); \]

• This is the same as calling the function directly:

   \[ y = \text{Foo}(x); \]
Function Pointers
Passing a Function to a Function

Example: Understanding the Mechanism

```c
int x;
int Foo(int a, int b);       // Function prototype

// Function definition with function pointer parameter
int Foobar(int a, int b, int (*fp)(int, int))
{
    return fp(a, b);         // Call function passed by pointer
}

void main(void)
{
    x = Foobar(5, 12, Foo);  // Pass address of foo
}
```
Function Pointers

Passing a Function to a Function

Example: Evaluate a definite integral (approximation)

```c
float Integrate(float from, float to, float (*f)(float))
{
    float sum = 0.0;
    float x;
    int n;

    // Evaluate integral{a,b} f(x) dx
    const float span = to - from;
    for (n = 0; n <= 100; n++) {
        x = ((n / 100.0) * span) + from;
        sum += (f(x) * span) / 101.0;
    }

    return sum;
}
```

Adapted from example at: http://en.wikipedia.org/wiki/Function_pointer
Function Pointers
Passing a Function to a Function

Example: Generic LinkedList

typedef struct ListItem {
    struct ListItem *previousItem;
    struct ListItem *nextItem;
    void *data;
} ListItem;

int LinkedListPrint(const ListItem *list,
    void (*Print)(const ListItem *));

int LinkedListSort(ListItem *list,
    const ListItem *(*Compare)(const ListItem *));
Void pointers
Void pointers

**Definition**

Void pointers are pointers that can hold a pointer to any type of data.

- Cannot be dereferenced
  - The size of the data cannot be inferred
  - Needs to be cast first
- Cannot point to functions
- Are big enough to store any pointer
Void pointers

Implicit casting

• Implicitly cast to other pointer types

Example

Node *node = malloc(sizeof(Node));

int *node = malloc(sizeof(Node));

void *node = malloc(sizeof(Node));
Void pointers
Dereferencing

• Void pointers cannot be dereferenced

Example

```c
void *node = malloc(sizeof(Node));

node->data = 'a';
```
Void pointers

Dereferencing

- Void pointers cannot support pointer math
  - No associated size

**Example**

```c
void *node = malloc(2 * sizeof(Node));

(node + 1)->data = 'b';
```
Variable-length arguments
Variable-length arguments

Syntax

```
type Name(type1 arg1, ..., type_n arg_n, ...);
```

- Requires at least one named argument
- ... states that the number and types the arguments may vary
  - It must be the last argument
- `<stdarg.h>` defines macros for iterating through all arguments
Variable-length arguments

Argument count

• No way to know how many arguments
• Solutions:
  – A count argument
  – A sentinel value
  – Use a formatting string like printf/scanf
Variable-length arguments

Iteration: Count argument

Example

```c
#include <stdarg.h>
int AllSum(int count, ...) {
    // Declare our argument pointer
    va_list argPtr;

    // Grab the first argument
    va_start(argPtr, count);

    int sum = 0;
    for (; count > 0; --count) {
        sum += va_arg(argPtr, int);
    }
    va_end(argPtr);

    return sum;
```
Variable-length arguments

Iteration: Sentinel value

Example

```c
#include <stdarg.h>
int AllSum(int arg1, ...)
{
    // Declare our argument pointer
    va_list argPtr;

    // Grab the first argument
    va_start(argPtr, arg1);

    int arg, sum = 0;
    for (arg = arg1; arg; arg = va_arg(argPtr, int)) {
        sum += arg;
    }
    va_end(argPtr);

    return sum;
}
```
Writing programs

Return values
Arguments
Writing Programs

Return values

- In a standard C environment, there is an Operating System
- Programs are started, execute, and end within the OS
- The return value allows for a program to return a code indicating its operation
- Most useful when writing daemons or programs that are not directly executed by the user
Writing Programs

Return values

- Returning 0 indicates successful operation
- Returning non-zero indicates error

Example

```c
int main(void)
{
    return 0;
}
```
Writing Programs
Return values

• `<stdlib.h>` defines `EXIT_SUCCESS` and `EXIT_FAILURE`

Example

```c
int main(void)
{
    return EXIT_SUCCESS;
}
```
Writing Programs

Return values

Syntax

```c
void exit(int status);
```

- Defined in `<stdlib.h>`

Example

```c
int main(void)
{
    exit(EXIT_FAILURE);

    return EXIT_SUCCESS;
}
```
Writing Programs

Program arguments

- Programs can take a variable number of arguments
  - Just like functions
- The number of arguments is known
- Only makes sense in a multi-process environment
  - Doesn't work with XC32
Writing Programs

Program arguments

**Syntax**

```c
int main(int argc, char *argv[]);
```

- Arguments are passed as strings
- First argument is the program name

**Example**

```
ls -hal ~

mkdir .ssh

ln -s ~/Dropbox/config/.ssh .ssh
```
Writing Programs

Program arguments

`ln -s ~/Dropbox/config/.ssh .ssh`

```
4
argc
argv
```

Syntax

```c
int main(int argc, char *argv[]);`
```
Writing Programs

Example: Output all program arguments

```c
int main(int argc, char *argv[]) {
    int i;
    for (i = 0; i < argc; ++i) {
        printf("%s ", argv[i]);
    }

    return EXIT_SUCCESS;
}
```
CMPE-013/L

Morse Decoder Lab

Maxwell James Dunne
Spring 2016
International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.
typedef struct Node {
    struct Node *leftChild;
    struct Node *rightChild;
    char data;
} Node;
Tree

Node *TreeCreate(int level, const char *data)

"1 2 3 4 5 6 7 8"

Levels

LC = TreeCreate(5, "1 2 3")
RC = TreeCreate(5, "4 5 6 7 8")
Morse Code

char MorseDecode(MorseChar in)

Dot

Dash

E

T

NULL

I

M

S

N

H

G

V

Z

F

Q

U

O

R

X

W

C

L

Y

P

0

J

1

B

2

K

3

D

4

N

5

X

6

C

7

Y

8

Z

9

Q

0
Node

curNode = curNode \rightarrow \text{Left child};
Main:

while (E)

    case D:

        Me = Me

        interrupt

        n

    switch (Me)