Recitation 9/11

Welcome to your first recitation!

Vote Garrett for TA of the week!

Logistics

- Survey 00 DUE TONIGHT!
- hw 01 due Friday (2 more days)
- Check-in 02 due Tuesday (6 more days)

Recitation Overview

– Pointers / C Practice

– Bits & Binary

Pointers!

- Given the following code, what will be printed?
  - HINT: I recommend drawing this out with boxes & arrows

(answer on next slide)

```c
void foo(int *x, int *y, int *z) {
  x = y;
  *x = *z;
  *z = 37;
}

int main(int argc, char *argv[]) {
  int x = 5, y = 22, z = 42;
  foo(&x, &y, &z);
  printf("%d, %d, %d\n", x, y, z);
  return 0;
}
```

- Given the following code, what will be printed?
  - Answer: 5, 42, 37

```c
void foo(int *x, int *y, int *z) {
  x = y;
  *x = *z;
  *z = 37;
}

int main(int argc, char *argv[]) {
  int x = 5, y = 22, z = 42;
  foo(&x, &y, &z);
  printf("%d, %d, %d\n", x, y, z);
  return 0;
}
```

Output Parameters

- Consider the following function:
  - Will the user get 5 as output?

```
void get_five(int out) {
  out = 5;
}

int main() {
  int x;
  get_five(x);
}
```

- **No!** you need to use a pointer so that the function can access the integer owned by the caller

```
void get_five(int *out) {
  *out = 5;
}

int main() {
  int x;
  get_five(&x);
}
```

Bits & Binary!

Base 10

$10^0 = 1$

$10^1 = 10$

$10^2 = 100$

$10^3 = 1,000$

$10^4 = 10,000$

Ex: 4,253

$3 \text{ x } 10^0$

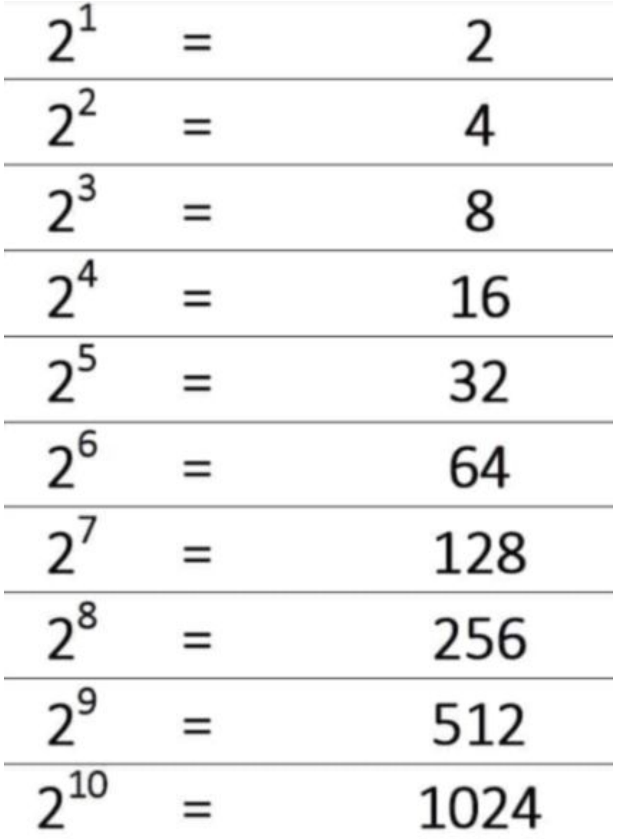
$5 \text{ x } 10^1$

$2 \text{ x } 10^2$

$4 \text{ x } 10^3$

$0 \text{ x } 10^4$

Base 2 - Binary



2^1	=	2
2^2	=	4
2^3	=	8
2^4	=	16
2^5	=	32
2^6	=	64
2^7	=	128
2^8	=	256
2^9	=	512
2^{10}	=	1024

Ex: 4,253 = b1000010011101

1×2^0

0×2^1

1×2^2

1×2^3

1×2^4

0×2^5

0×2^6

1×2^7

0×2^8

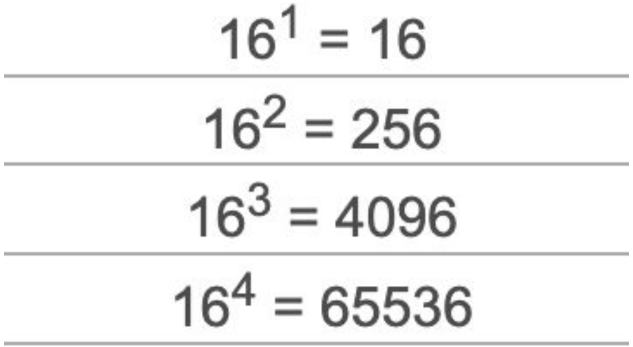
0×2^9

0×2^{10}

0×2^{11} (2048)

1×2^{12} (4096)

Base 16 - Hexadecimal

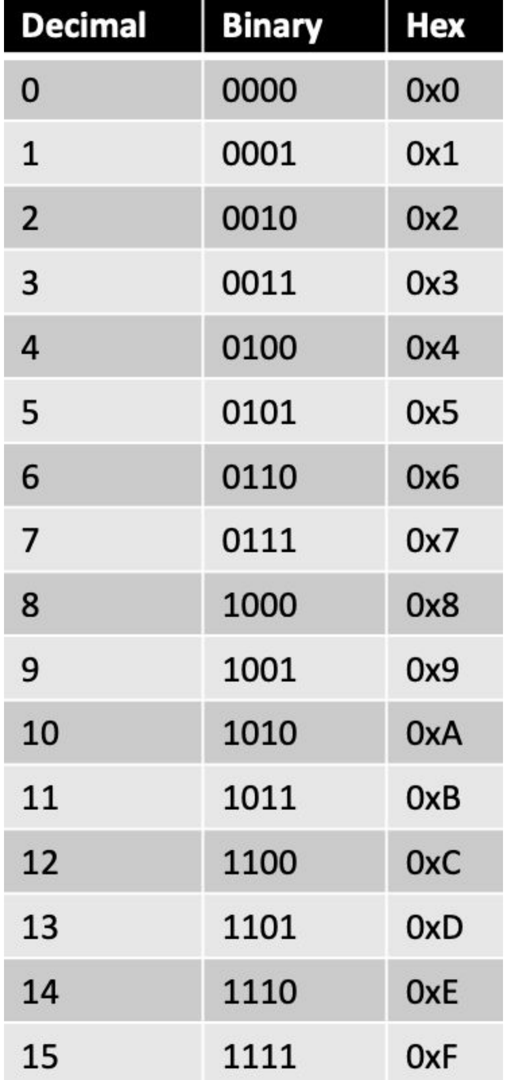


$16^1 = 16$

$16^2 = 256$

$16^3 = 4096$

$16^4 = 65536$



Decimal	Binary	Hex
0	0000	0x0
1	0001	0x1
2	0010	0x2
3	0011	0x3
4	0100	0x4
5	0101	0x5
6	0110	0x6
7	0111	0x7
8	1000	0x8
9	1001	0x9
10	1010	0xA
11	1011	0xB
12	1100	0xC
13	1101	0xD
14	1110	0xE
15	1111	0xF

Binary to Hex

Ex: 4,253 = b1000010011101 = 0x????

1. Starting from right, chunk the bits in groups of 4

    1 0000 1001 1101

2. Optional step: pad the left with 0’s until you have groups of 4

    0001 0000 1001 1101

3. Convert each 4-bit chunk to its hex equivalent

    109D

Answer: 4,253 = 0x109D

Binary Question

Base 2:
Base 10: 108
Base 16:

Base 2:
Base 10:
Base 16: 3D

Base 2: 100010
Base 10:
Base 16:

Base 2:
Base 10: 175
Base 16:

*assume unsigned binary

Binary Question

Base 2: **1101100**
Base 10: 108
Base 16: **70**

Base 2: **00111101**
Base 10: **61**
Base 16: 3D

Base 2: 100010
Base 10: **34**
Base 16: **22**

Base 2: **10101111**
Base 10: 175
Base 16: **AF**

*assume unsigned binary

2’s Complement

- Signed integers (can represent positive or negative values)
- Negative Numbers - 1 Most Significant Bit
- Positive Numbers - 0 Most Significant Bit
- Negate binary numbers : Invert (1’s turn to 0’s and 0’s turn to 1’s) -> Plus 1

Practice - Use 8 bits

- Base 10 : - 83
- Base 2:

- Base 10 :
- Base 2: 1100 1000

- Base 10:
- Base 2: 0111 1101

- Base 10: 14
- Base 2:

2’s Complement

- Signed integers (can represent positive or negative values)
- Negative Numbers - 1 Most Significant Bit
- Positive Numbers - 0 Most Significant Bit
- Negate binary numbers : Invert (1’s turn to 0’s and 0’s turn to 1’s) -> Plus 1

Practice - Use 8 bits

- Base 10 : - 83
- Base 2: 1010 1101

- Base 10: 125
- Base 2: 0111 1101

- Base 10 : - 56
- Base 2: 1100 1000

- Base 10: 14
- Base 2: 0000 1110

Bit Operators

& - Bitwise And

1 & 1 = 1

1 & 0 = 0

0 & 1 = 0

0 & 0 = 0

| - Bitwise Or

1 | 1 = 1

1 | 0 = 1

0 | 1 = 1

0 | 0 = 0

^ - Xor

1 ^ 1 = 0

1 ^ 0 = 1

0 ^ 1 = 1

1 ^ 1 = 0

<< - Left Shift

1 << 1 = b10

1 << 2 = b100

1 << 3 = b1000

>> - Right Shift

2 >> 1 = 1

2 >> 2 = 0

2 >> 3 = 0

Notes:

2 >> -1 = undefined!

2 << -1 = undefined!

Logical (Boolean) Operators

&& - Logical And

T && T = T

T && F = F

F && T = F

F && F = F

|| - Logical Or

T || T = T

T || F = T

F || T = T

F || F = F

! - Logical Not

!T = F

!F = T

Isolate the bit at index i (i+1th bit from the right)

Given x = number and i = index of bit from the right

1. Zero-out the lower i bits: x >> i, then x << i
2. Create and apply a mask - but there are multiple ways to do this!
   a. Goal: mask is a number with i+1 ones on the right, and everything else is zero

Method 1:

1. M = 0
2. M1 = ~M (now all bits are 1's)
3. M2 = M1 >> (i+1)
4. M3 = M2 << (i+1) (lower bits are zeroed out, add 1 to include bit at index i)
5. M4 = ~M3 (flip the bits; now left bits are zeroed out while right bits are 1)

Method 2:

1. N = 1 << (i+1) (1 with i+1 zero's after)
2. N1 = N - 1 (all i+1 zero's are turned to 1's and, everything to the left is now zero)

Boolean logic tricks

- What is the binary representation of the smallest 2C 16-bit integer?
- How to get -1 in binary without using - sign?
- !!x is not x
- Bitmask: &(~0), &0, & 0xFF
- -1 + 1 = 0
- x ^ 0; x ^ -1
- Setting a bit x | (1 << 2);
- Clearing a bit x & ~(1 << 2);
- Flip a bit: x ^ (1 << 2);

C practice!

Yes or No?

1. Is NULL the same as 0?
2. Is the ***sizeof*** operator evaluated at compile time?
3. Is the size of a pointer always the same as the size of int on any system?
4. Is it possible to use bitwise operators on pointers in C?
5. Does shifting an integer more than its bit width result in undefined behaviors?
6. Is C statically typed?
7. Is C strongly typed?

Java versus C. What's the difference?

- Memory access: C does not enforce array size restriction -> will not throw out-of-bounds error if you try to access a[5] on a size 3 array!
    - C will not store the length of the array ANYWHERE in its memory
- Java always passes its variables as references (pointers to the object). In C, you have to make this explicit. You need to tell the code that you are passing in a pointer, otherwise it will just copy the value stored in the pointer (pass by value).
- String type (Java) versus char* (C)
- Java is compiled into bytecode, which is platform independent for any machine that has Java Virtual Machine (JVM) installed; For C, however, it is compiled down to machine code, which depends on platform-specific compilers.

- In summary: C being more "low level" means that **you have to be more precise and explicit** when designing program procedures - a lot of things (such as memory management) are not "handled for you" like in Java

Check out Daniel's awesome ed post https://edstem.org/us/courses/62374/discussion/5201975 (C vs Java, #36)

It’s not you, it’s C

It’s not your fault, it’s seg fault.