CS 31: Intro to Systems C Programming

L04: Data representation

Vasanta Chaganti & Kevin Webb

Swarthmore College

September 14, 2023

Announcements

- HW1 is due now.
- Lab 1 is due Today (Thursday, 11.59 PM)
- Clickers will count for credit from this week

Reading Quiz

- Note the red border!
- 1 minute per question
- No talking, no laptops, phones during the quiz

Check your frequency:

- Iclicker2: frequency AA
- Iclicker+: green light next to selection

For new devices this should be okay,
For used you may need to reset frequency

Reset:

1. hold down power button until blue light flashes (2secs)
2. Press the frequency code: AA vote status light will indicate success

Agenda

Data representation

- number systems + conversion
- data types, storage
- sizes, representation
- signedness

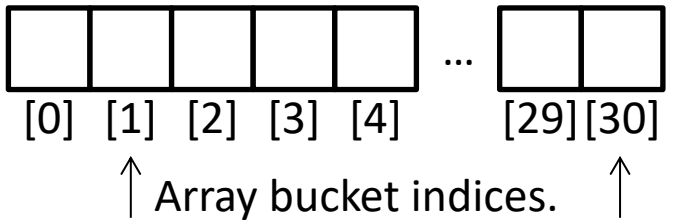
Arrays and Strings

C’s support for collections of values

- Array buckets store a single type of value

Array Characteristics

```
int january_temps[31];
```

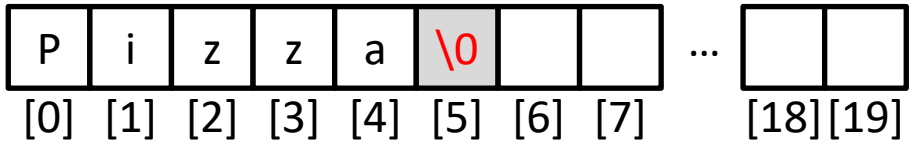


- Indices start at 0!
- The index refers to an offset from the start of the array
- Add temperature reading for January 5th?

Characters and Strings

- A character (type `char`) is numerical value that holds one letter.
- A string is a memory block containing characters, one after another, with a null terminator (numerical 0) at the end.

  char name[20] = “Pizza”; //C appends the null terminator for you in this declaration!



- What is the minimum size of a char array that we must declare to hold the string “Swarthmore” ?

  char school[11] = “Swarthmore”;

What will this print?

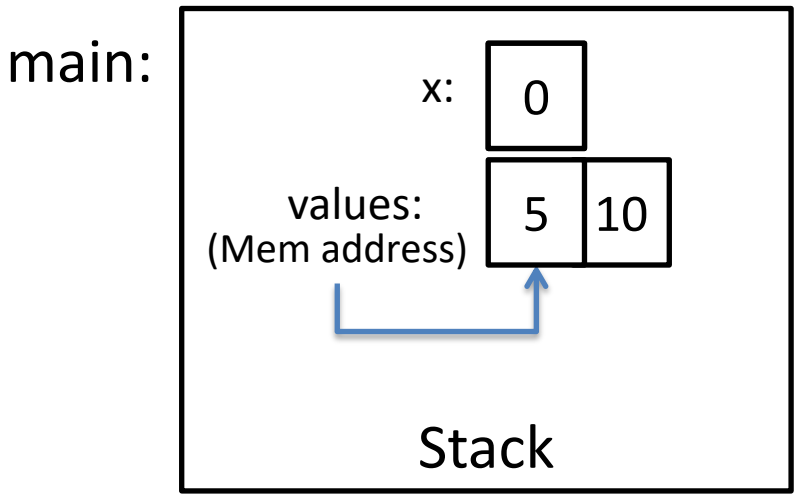
```
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```

A. 0, 5, 8
B. 0, 5, 10
C. 1, 0, 8
D. 1, 5, 8
E. 1, 5, 10

Hint: What does the name of an array mean to the compiler?

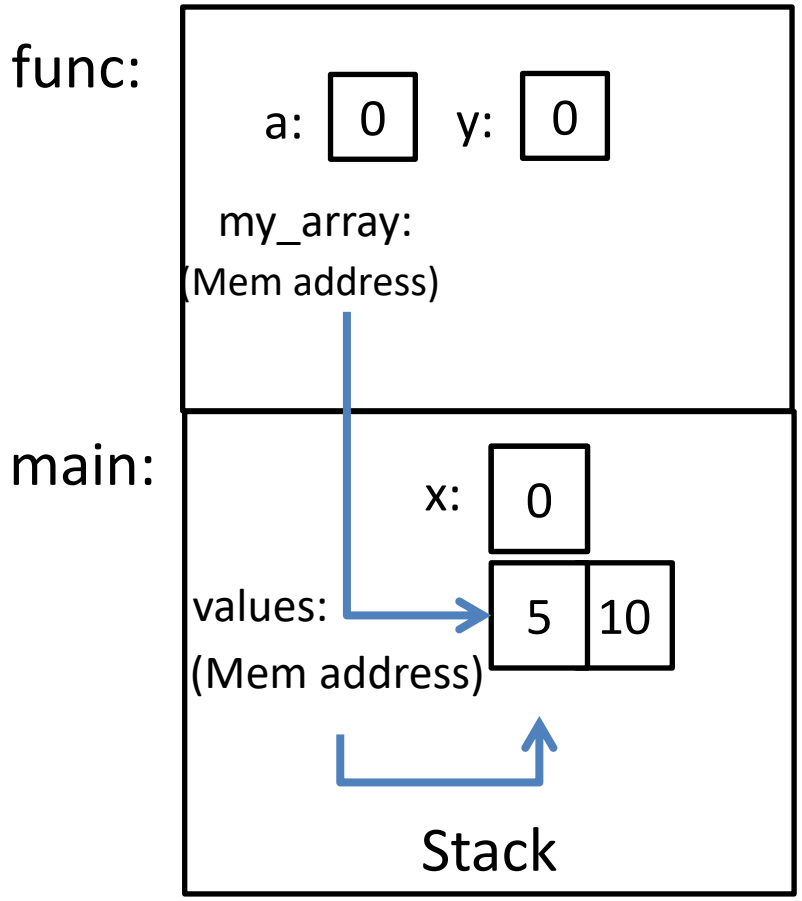
What will this print?

```
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```



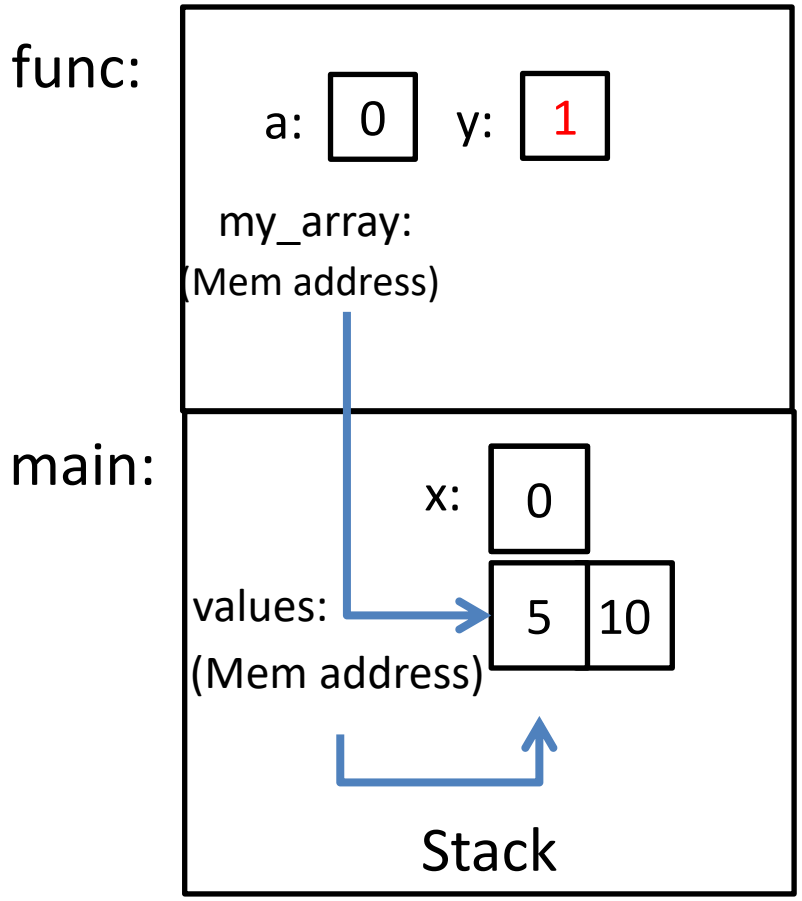
What will this print?

```
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```



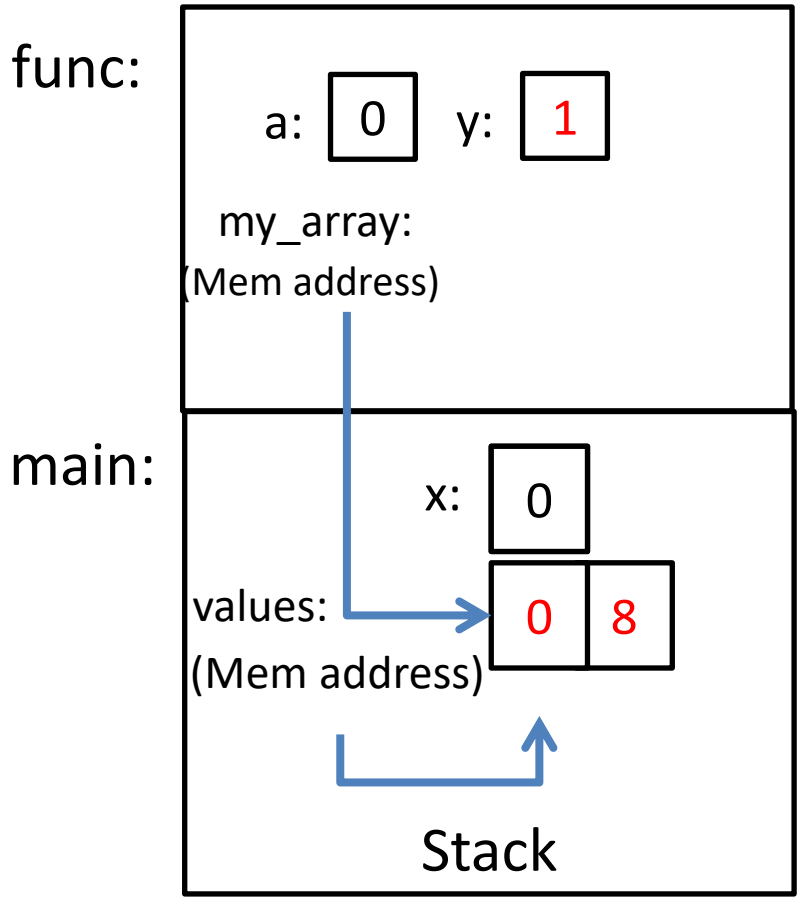
What will this print?

```
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```



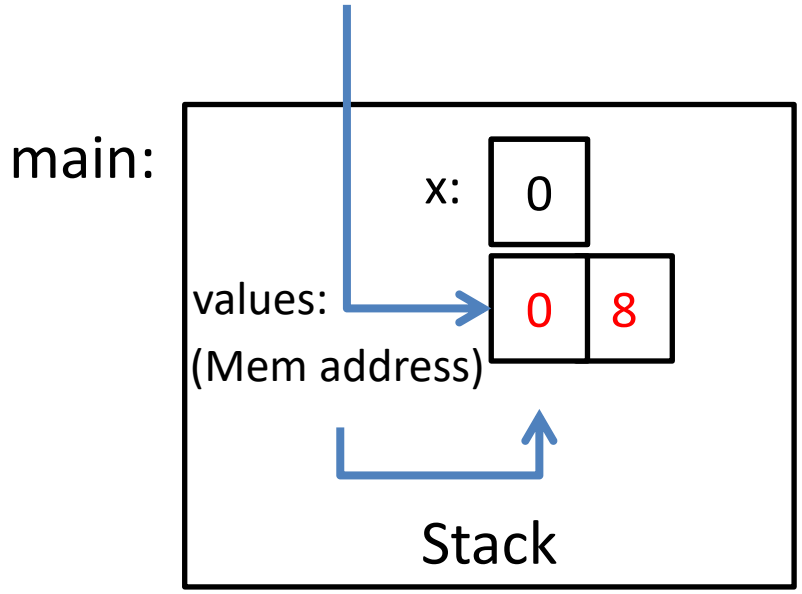
What will this print?

```
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```



What will this print?

```
int func(int a, int y, int my_array[]) {
    y = 1;
    my_array[a] = 0;
    my_array[y] = 8;
    return y;
}

int main() {
    int x;
    int values[2];

    x = 0;
    values[0] = 5;
    values[1] = 10;

    x = func(x, x, values);

    printf("%d, %d, %d", x, values[0], values[1]);
}
```



structs

- Treat a collection of values as a single type:
  - C is not an object oriented language, no classes
  - A `struct` is similar to the data part of a class

- Rules:
  1. Define a new struct type outside of any function
  2. Declare variables of the new struct type
  3. Use dot notation to access the field values of a struct variable

Struct Example

Suppose we want to represent a *student* type.

```
struct student {
    char name[20];
    int grad_year;
    float gpa;
};
// Variable bob is of type struct student
struct student bob;
// Set name (string) with strcpy()
strcpy(bob.name, “Robert Paulson”);
bob.grad_year = 2019;
bob.gpa = 3.1;

printf(“Name: %s, year: %d, GPA: %f”, bob.name, bob.grad_year, bob.gpa);
```

Arrays of Structs

```
struct student {
    char name[20];
    int grad_year;
    float gpa;
};
//create an array of struct students!
struct student classroom[50];

strcpy(classroom[0].name, “Alice”);
classroom[0].grad_year = 2023
classroom[0].gpa = 4.0;
```

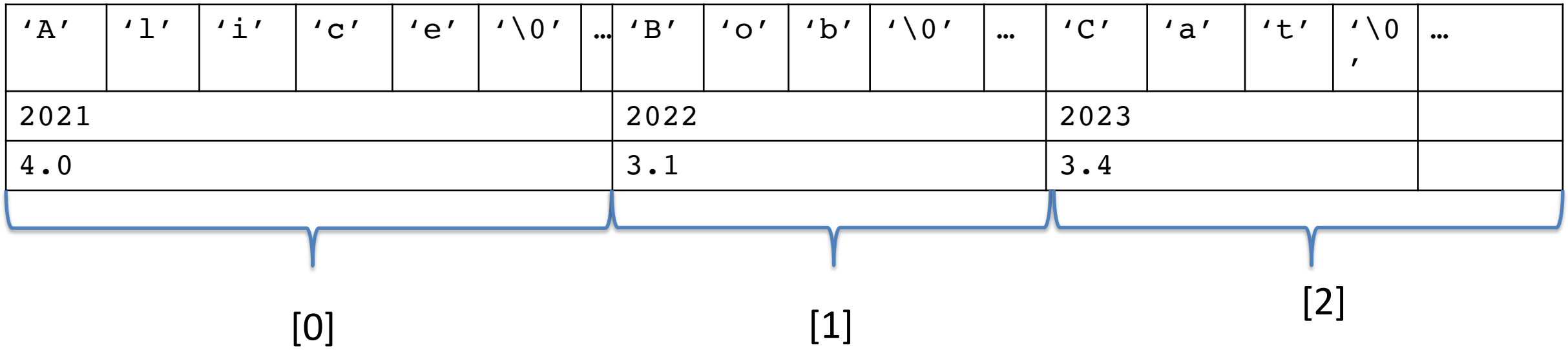
```
// With a loop, create an army of Alice clones!
int i;
for (i = 0; i < 50; i++) {
        strcpy(classroom[i].name, “Alice”);
        classroom[i].grad_year = 2023;
        classroom[i].gpa = 4.0;
}
```

Arrays of Structs

```
struct student classroom[3];

strcpy(classroom[0].name, “Alice”);
classroom[0].grad_year = 2021;
classroom[0].gpa = 4.0;

strcpy(classroom[1].name, “Bob”);
classroom[1].grad_year = 2022;
classroom[1].gpa = 3.1

strcpy(classroom[2].name, “Cat”);
classroom[2].grad_year = 2023;
classroom[2].gpa = 3.4
```

Array of Structs: Layout in Memory

`classroom: array of structs`



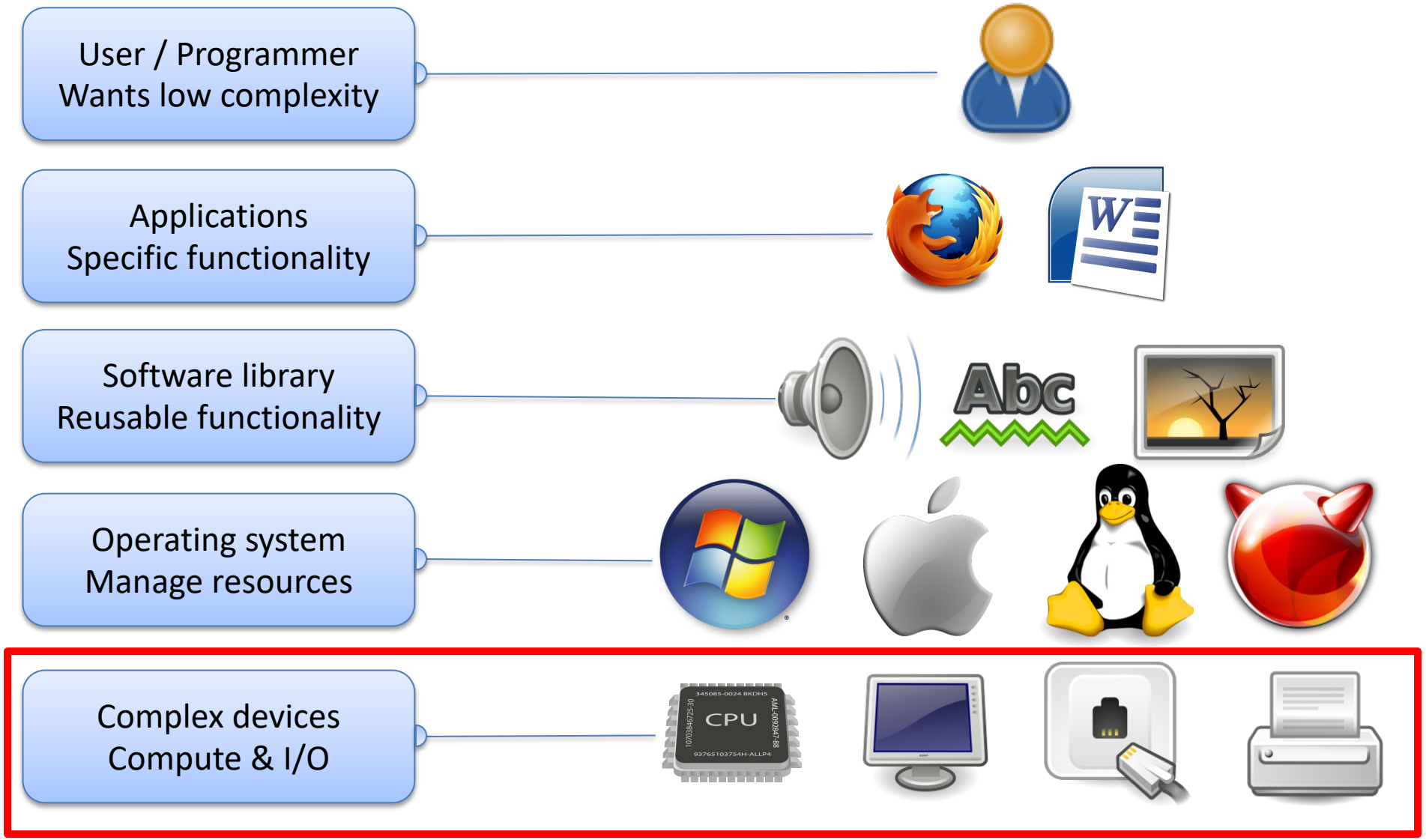
Q1 Discussion Block 2 of Worksheet

Structs

```c
#include <stdio.h>

struct personT {
  char name[32];
  int  age;
  float heart_rate;
};

int main(void) {
  struct personT  p1;
  struct personT  people[40];
  return 0;
}
```

expression	type
p1	
p1.name	
p1.heart_rate	
people	
people[0]	
people[0].name	
people[0].name[3]	

Abstraction



Data Storage

- Lots of technologies out there:
  - Magnetic (hard drive, floppy disk)
  - Optical (CD / DVD / Blu-Ray)
  - Electronic (RAM, registers, …)
- Focus on electronic for now
  - We'll see (and build) digital circuits soon
- Relatively easy to differentiate two states
  - Voltage present
  - Voltage absent

Bits and Bytes

- Bit: a 0 or 1 value (binary)
  - HW represents as two different voltages
    - 1: the presence of voltage (high voltage)
    - 0: the absence of voltage (low voltage)

- Byte: 8 bits, the smallest addressable unit

Memory:	01010101	10101010	00001111	…
(address)	[0]	[1]	[2]	…

- Other names:
  - 4 bits: Nibble
  - “Word”: Depends on system, often 4 bytes

Files

Sequence of bytes… nothing more, nothing less

Binary Digits (BITs)

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.

How many unique values can we represent with 9 bits? Why?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.

How many values?

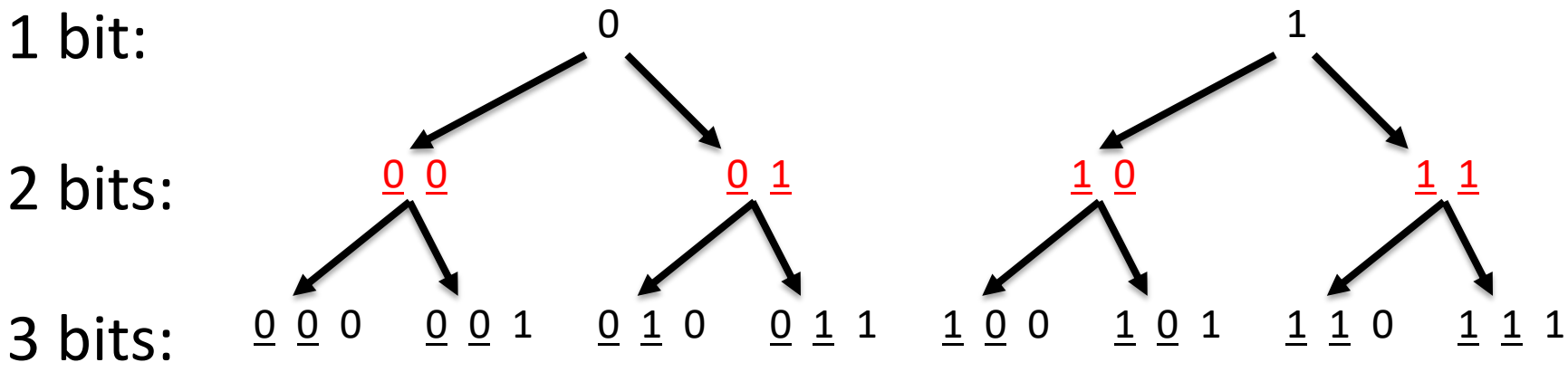
1 bit: 0 1

How many values?

1 bit:

2 bits:

How many values?

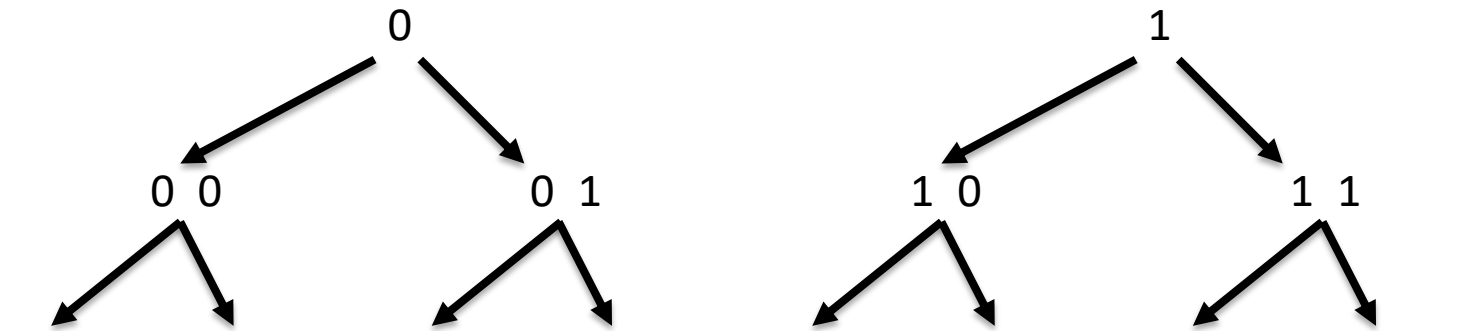


How many values?

1 bit:

2 bits:

3 bits:



4 bits:

0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 16 values
0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1

1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1
1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1

N bits: 2^N values

C types and their (typical!) sizes

- 1 byte: char, unsigned char
- 2 bytes: short, unsigned short
- 4 bytes: int, unsigned int, float
- 8 bytes: long long, unsigned long long, double
- 4 or 8 bytes: long,

WARNING: These sizes are **NOT** a guarantee. Don't always assume that every system will use these values!

```
unsigned long v1;
short s1;
long long ll;

// prints out number of bytes
printf("%lu %lu %lu\n", sizeof(v1), sizeof(s1), sizeof(ll));
```

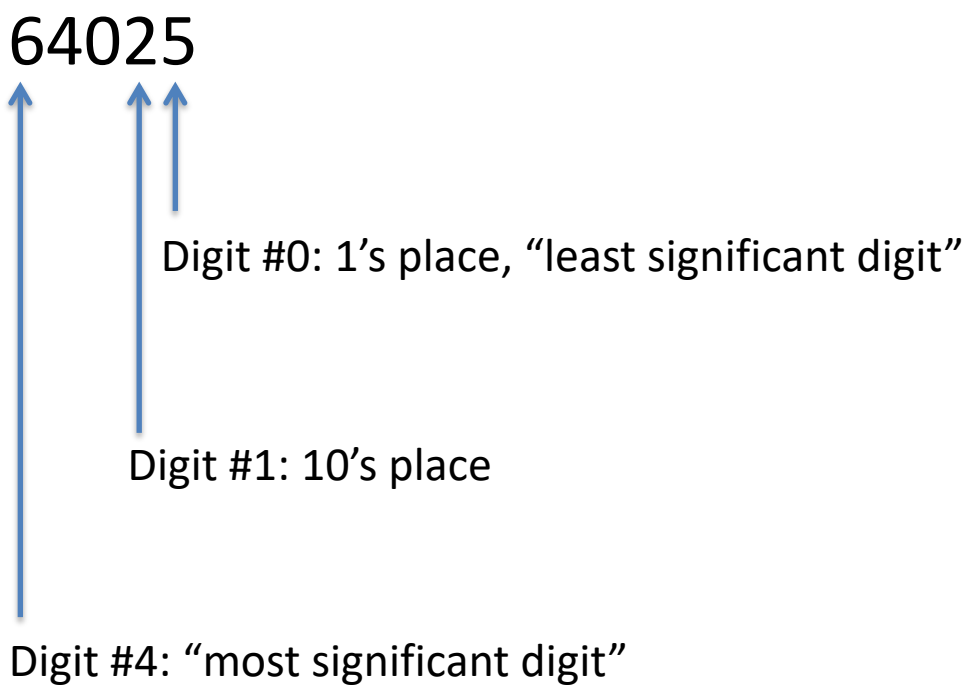
How do we use this storage space (bits) to represent a value?

Let's start with what we know…

- Digits 0-9
- Positional numbering
- Digits are composed to make larger numbers
- Known as Base 10 representation

Decimal number system (Base 10)

- Sequence of digits in range [0, 9]



Decimal: Base 10

A number, written as the sequence of N digits,

$$d_{n-1} \ldots d_2 \, d_1 \, d_0$$

where d is in {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, represents the value:

$$[d_{n-1} * 10^{n-1}] + [d_{n-2} * 10^{n-2}] + \ldots + [d_1 * 10^1] + [d_0 * 10^0]$$

64025 =

$6 * 10^4 + 4 * 10^3 + 0 * 10^2 + 2 * 10^1 + 5 * 10^0$

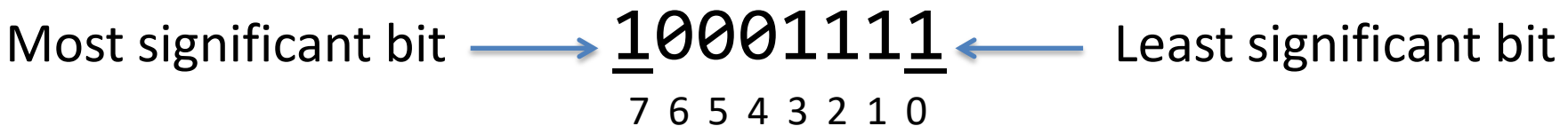
60000 + 4000 + 0 + 20 + 5

Binary: Base 2

- Used by computers to store digital values.
- Indicated by prefixing number with **0b**
- A number, written as the sequence of N digits, $d_{n-1}…d_2d_1d_0$, where d is in {0,1}, represents the value:

$$[d_{n-1} * 2^{n-1}] + [d_{n-2} * 2^{n-2}] + ... + [d_2 * 2^2] + [d_1 * 2^1] + [d_0 * 2^0]$$

Converting Binary to Decimal



Representation: $1 \times 2^7 + 0 \times 2^6 \ldots + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$128 + \quad + 8 + 4 + 2 + 1$

$10001111 = 143$

Hexadecimal: Base 16

Indicated by prefixing number with **0x**

A number, written as the sequence of N digits,

$$d_{n-1}\ldots d_2 d_1 d_0,$$

where d is in {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}, represents:

$$[d_{n-1} * 16^{n-1}] + [d_{n-2} * 16^{n-2}] + \ldots + [d_2 * 16^2] + [d_1 * 16^1] + [d_0 * 16^0]$$

Generalizing: Base b

The meaning of a digit depends on its position in a number.

A number, written as the sequence of N digits,

$$d_{n-1} \ldots d_2\, d_1\, d_0$$

in base b represents the value:

$$[d_{n-1} * b^{n-1}] + [d_{n-2} * b^{n-2}] + \ldots + [d_2 * b^2] + [d_1 * b^1] + [d_0 * b^0]$$

Base 10: $[d_{n-1} * 10^{n-1}] + [d_{n-2} * 10^{n-2}] + \ldots + [d_1 * 10^1] + [d_0 * 10^0]$

Other (common) number systems.

- Base 2: How data is stored in hardware.
- Base 8: Used to represent file permissions.
- Base 10: Preferred by people.
- Base 16: Convenient for representing memory addresses.
- Base 64: Commonly used on the Internet, (e.g. email attachments).

It's **all** stored as binary in the computer.

Different representations (or visualizations) of the same information!

Q1 Discussion Block 2 of Worksheet

What is the value of 0b110101 in decimal?

A number, written as the sequence of N digits $d_{n-1}...d_2d_1d_0$ where d is in {0,1}, represents the value:

$$[d_{n-1} * 2^{n-1}] + [d_{n-2} * 2^{n-2}] + ... + [d_2 * 2^2] + [d_1 * 2^1] + [d_0 * 2^0]$$

A. 26
B. 53
C. 61
D. 106
E. 128

What is the value of 0x1B7 in decimal?

$[d_{n-1} * 16^{n-1}] + [d_{n-2} * 16^{n-2}] + \ldots + [d_2 * 16^2] + [d_1 * 16^1] + [d_0 * 16^0]$

(Note: $16^2 = 256$)

A. 397
B. 409
C. 419
D. 437
E. 439

DEC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Important Point…

- You can represent the same value in a variety of number systems or bases.

- It's **all** stored as binary in the computer.
  - Presence/absence of voltage.

Hexadecimal: Base 16

- Fewer digits to represent same value
  - Same amount of information!
- Like binary, the base is power of 2
- Each digit is a “nibble”, or half a byte.

Each hex digit is a "nibble"

- One hex digit: 16 possible values (0-9, A-F)
- $16 = 2^4$, so each hex digit has exactly four bits worth of information.
- We can map each hex digit to a four-bit binary value.
  (helps for converting between bases)

Each hex digit is a “nibble”

Example value: 0x1B7

Four-bit value: 1

Four-bit value: B (decimal 11)

Four-bit value: 7

In binary: 0001 1011 0111

1 B 7

Converting Decimal -> Binary

- Two methods:
  - division by two remainder
  - powers of two and subtraction

Method 1: decimal value D, binary result b (b_i is ith digit):

```
i = 0
while (D > 0)
   if D is odd
          set b_i to 1
   if D is even
          set b_i to 0
   i++
   D = D/2
```

Example: Converting 105

idea: example: D = 105 b_0 = 1

Method 1: decimal value D, binary result b (b_i is ith digit):

```
i = 0
while (D > 0)
   if D is odd
          set b_i to 1
   if D is even
          set b_i to 0
   i++
   D = D/2
```

Example: Converting 105

idea: D, D = D/2 example: D = 105, D = 52 $b_0 = 1$, $b_1 = 0$

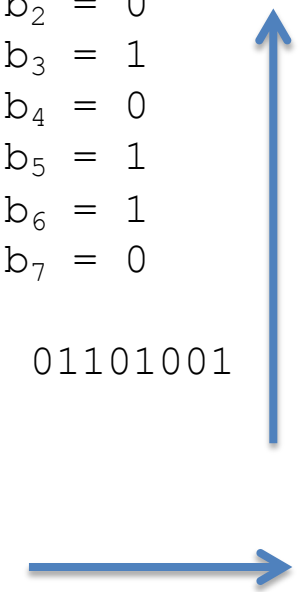
Method 1: decimal value D, binary result b (b_i is ith digit):

```
i = 0
while (D > 0)
   if D is odd
         set b_i to 1
   if D is even
         set b_i to 0
   i++
   D = D/2
```

Example: Converting 105

idea: D	example: D = 105	b_0 = 1
D = D/2	D = 52	b_1 = 0
D = D/2	D = 26	b_2 = 0
D = D/2	D = 13	b_3 = 1
D = D/2	D = 6	b_4 = 0
D = D/2	D = 3	b_5 = 1
D = D/2	D = 1	b_6 = 1
D = 0 (done)	D = 0	b_7 = 0

105 = 01101001



Method 2

- $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$
-

  To convert 105:
  - Find largest power of two that's less than 105 (64)
  - Subtract 64 (105 – 64 = 41), put a 1 in d_6
  - Subtract 32 (41 – 32 = 9), put a 1 in d_5
  - Skip 16, it's larger than 9, put a 0 in d_4
  - Subtract 8 (9 – 8 = 1), put a 1 in d_3
  - Skip 4 and 2, put a 0 in d_2 and d_1
  - Subtract 1 (1 – 1 = 0), put a 1 in d_0 (Done)

d_6	d_5	d_4	d_3	d_2	d_1	d_0
1	1	0	1	0	0	1

What is the value of 357 in binary?

8 7654 3210 → digit position

A. 1 0110 0011
B. 1 0110 0101
C. 1 0110 1001
D. 1 0111 0101
E. 1 1010 0101

$2^0 = 1, \quad 2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 8, \quad 2^4 = 16,$

$2^5 = 32, \quad 2^6 = 64, \quad 2^7 = 128, \quad 2^8 = 256$

What is the value of 357 in binary?

8 7654 3210 → digit position

A. 1 0110 0011
B. 1 0110 0101
C. 1 0110 1001
D. 1 0111 0101
E. 1 1010 0101

357 – 256 = 101
101 – 64 = 37
37 – 32 = 5
5 – 4 = 1

1	0	1	1	0	0	1	0	1
d_8	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

$2^0 = 1,\quad 2^1 = 2,\quad 2^2 = 4,\quad 2^3 = 8,\quad 2^4 = 16,$

$2^5 = 32,\quad 2^6 = 64,\quad 2^7 = 128,\quad 2^8 = 256$

So far: Unsigned Integers

With N bits, can represent values: 0 to 2^n-1

We can always add 0's to the front of a number without changing it:

10110= 010110 = 00010110 = 0000010110

So far: Unsigned Integers

With N bits, can represent values: 0 to 2^n-1

- 1 byte: char, unsigned char
- 2 bytes: short, unsigned short
- 4 bytes: int, unsigned int, float
- 8 bytes: long long, unsigned long long, double
- 4 or 8 bytes: long, unsigned long

Unsigned Integers

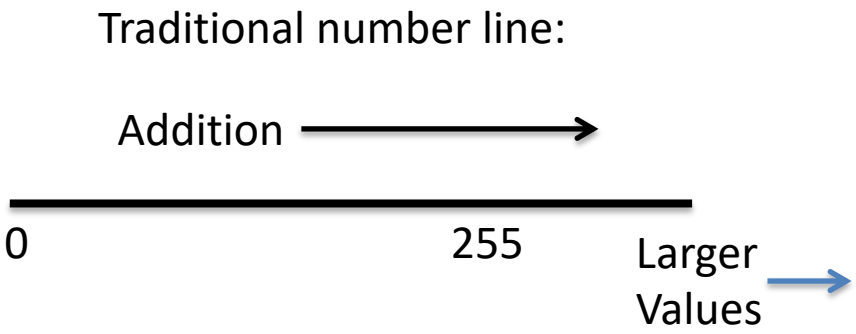
- Suppose we had one byte
  - Can represent 2^8 (256) values
  - If unsigned (strictly non-negative): 0 – 255

252 = 11111100

253 = 11111101

254 = 11111110

255 = 11111111



Unsigned Integers

Suppose we had one byte

- Can represent 2^8 (256) values
- If unsigned (strictly non-negative): 0 – 255

252 = 11111100

253 = 11111101

254 = 11111110

255 = 11111111

What if we add one more?

Car odometer “rolls over”.

Any time we are dealing with a finite storage space we cannot represent an infinite number of values!

Unsigned Integers

Suppose we had one byte

- Can represent 2^8 (256) values
- If unsigned (strictly non-negative):

  0 – 255

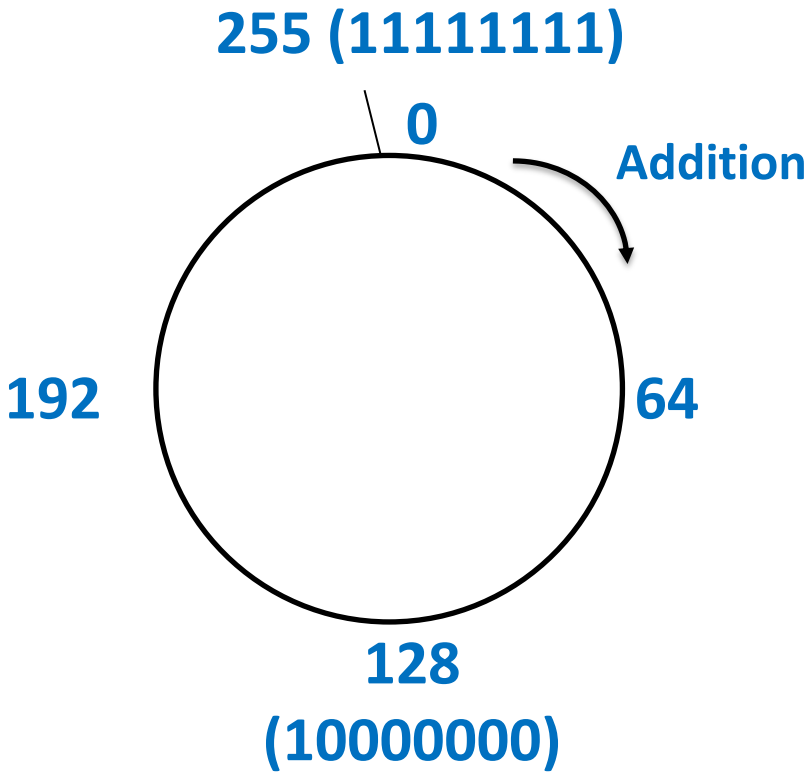
252 = 11111100

253 = 11111101

254 = 11111110

255 = 11111111

What if we add one more?



Modular arithmetic: Here, all values are modulo 256.

Unsigned Addition (4-bit)

- Addition works like grade school addition:

```
   1
   0110      6
+  0100   +  4
------    ----
   1010      10
```

Four bits give us range: 0 - 15

Unsigned Addition (4-bit)

- Addition works like grade school addition:

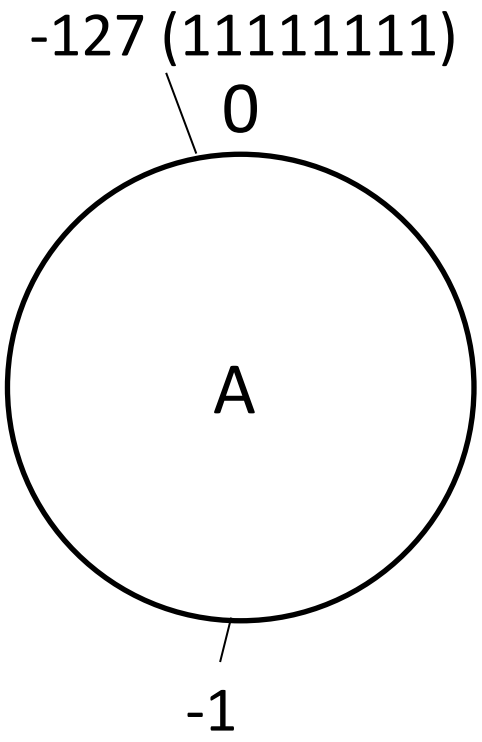
```
   1
  0110      6          1100     12
+ 0100    + 4        + 1010    +10
------    ---        ------    ---
  1010     10        1 0110      6
^no carry out      ^carry out
```

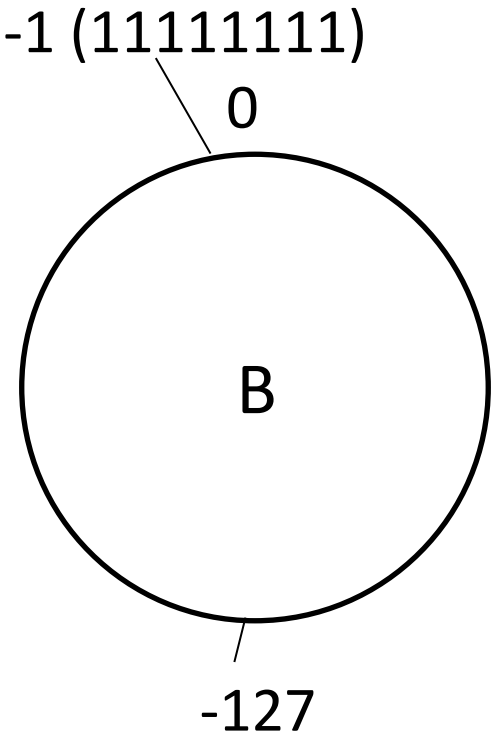
Four bits give us range: 0 - 15

Overflow!

Carry out is indicative of something having gone wrong when adding unsigned values

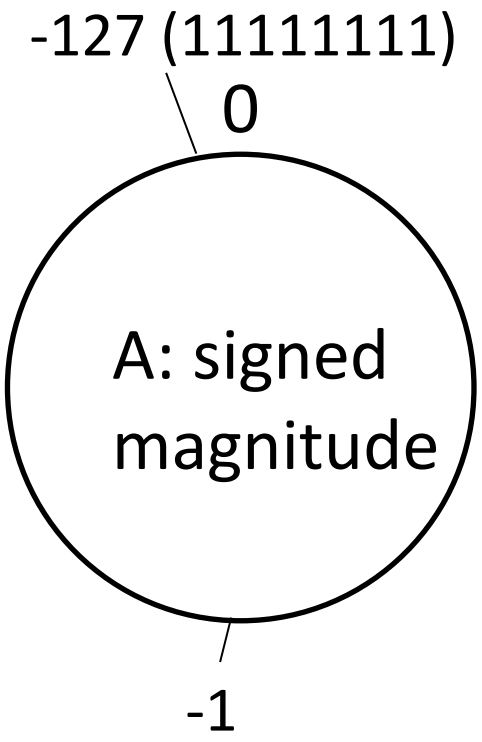
Suppose we want to support signed values (positive and negative) in 8 bits, where should we put -1 and -127 on the circle? Why?

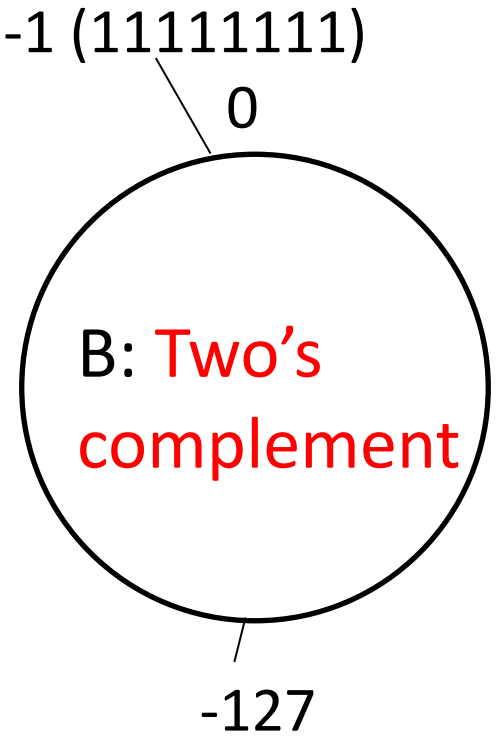




C: Put them somewhere else.

Suppose we want to support signed values (positive and negative) in 8 bits, where should we put -1 and -127 on the circle? Why?





C: Put them somewhere else.

Signed Magnitude Representation (for 4 bit values)

- One bit (usually left-most) signals:
  - 0 for positive
  - 1 for negative

For one byte:

1 = 00000001, -1 = 10000001

Pros: Negation (negative value of a number) is very simple!

For one byte:

0 = 00000000

What about 10000000?

Major con: Two ways to represent zero!

Two's Complement Representation (for four bit values)

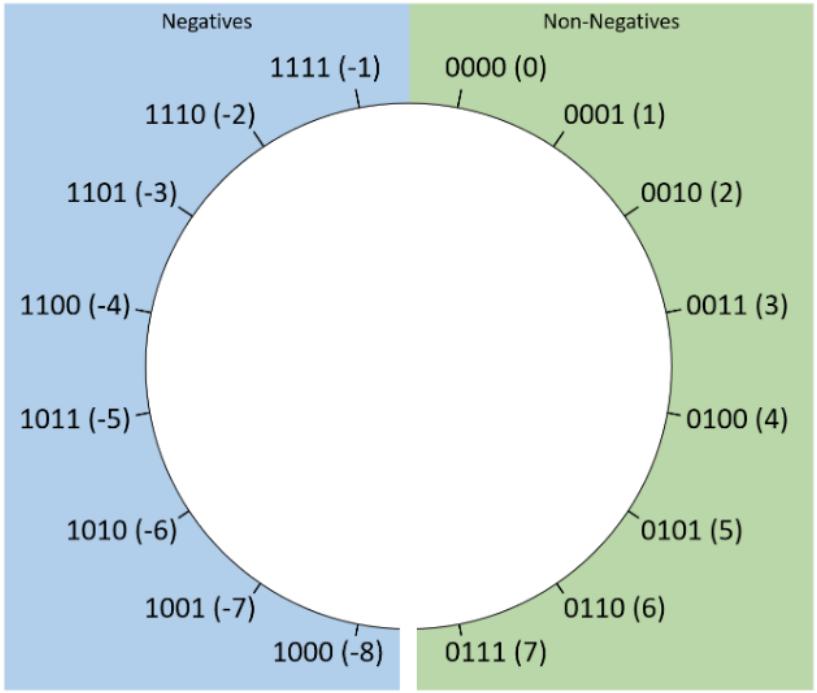
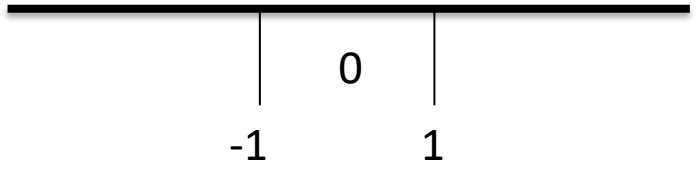


Figure 2. A logical layout of two's complement values for bit sequences of length four.

For an 8 bit range we can express 256 unique values:

- 128 non-negative values (0 to 127)
- 128 negative values (-1 to -128)

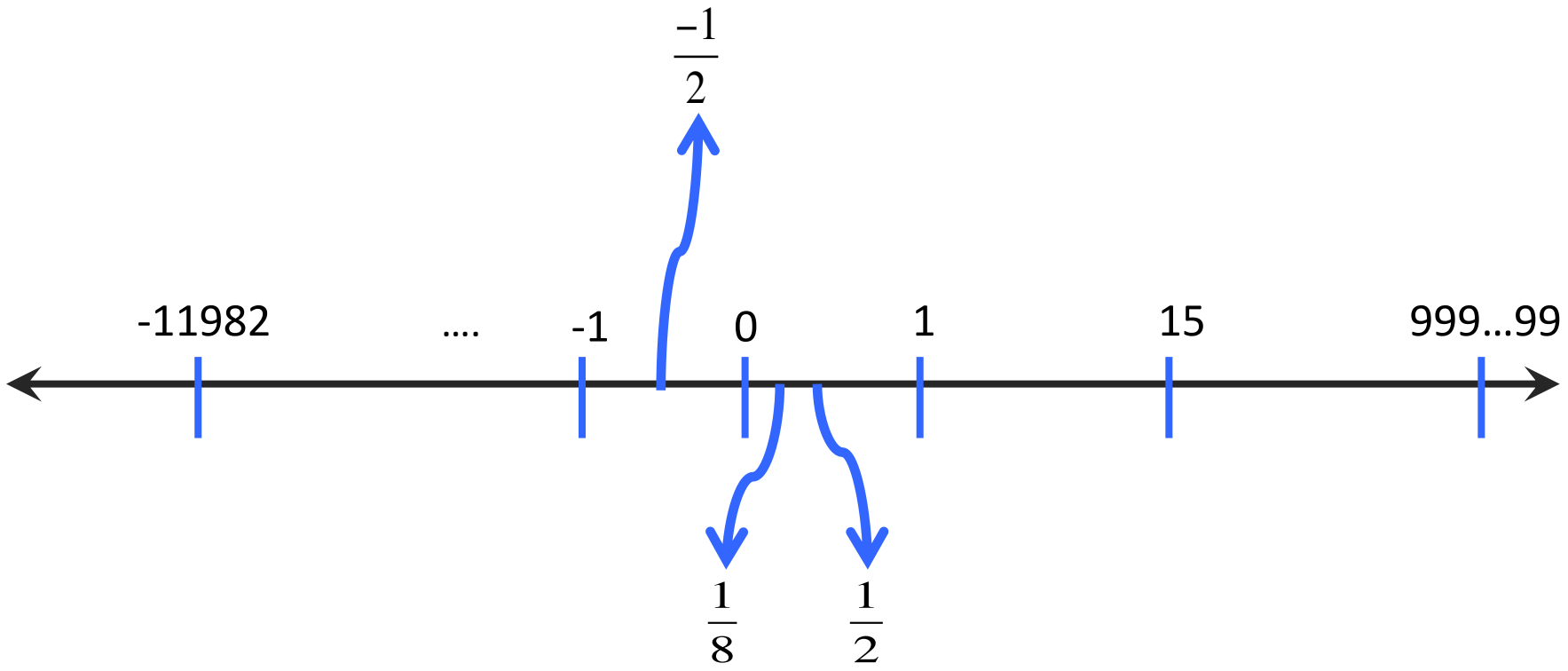
- Borrow nice property from number line:



Only one instance of zero!
Implies: -1 and 1 on either side of it.

Additional Info: Fractional binary numbers

How do we represent fractions in binary?



Additional Info: Representing Signed Float Values

- One option (used for floats, NOT integers)
  - Let the first bit represent the sign
  - 0 means positive
  - 1 means negative

- For example:
  - 0101 -> 5
  - 1101 -> -5

- Problem with this scheme?

A) no problem B) some numbers are represented twice,

C) some numbers have no representation

Additional Info: Floating Point Representation

1 bit for sign sign | exponent | fraction |
8 bits for exponent
23 bits for precision

value = $(-1)^{sign} * 1.fraction * 2^{(exponent-127)}$

let's just plug in some values and try it out

```
0x40ac49ba: 0 10000001   01011000100100110111010
     sign = 0 exp = 129   fraction = 2902458
```

$= 1*1.2902458*2^2 = 5.16098$

I don't expect you to memorize this

Summary

- Images, Word Documents, Code, and Video can represented in bits.
- Byte or 8 bits is the smallest addressable unit
- N bits can represent 2^N unique values
- A number is written as a sequence of digits: in the decimal base system
  - [dn * 10 ^ n] + [dn-1 * 10 ^ n-1] + ... + [d2 * 10 ^ 2] + [d1 * 10 ^ 1] + [d0 * 10 ^ 0]
  - For any base system:
  - [dn * b ^ n] + [dn-1 * b ^ n-1] + ... + [d2 * b ^ 2] + [d1 * b ^ 1] + [d0 * b ^ 0]
- Hexadecimal values (represent 16 values): {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
  - Each hexadecimal value can be represented by 4 bits. (2^4=16)
- A finite storage space we cannot represent an infinite number of values. For e.g., the max unsigned 8 bit value is 255.
  - Trying to represent a value >255 will result in an overflow.
- Two's Complement Representation: 128 non-negative values (0 to 127), and 128 negative values (-1 to -128).