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?

Array bucket indices.
Characters and Strings

• A character (type `char`) is a 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.

```c
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”? 

```c
```
```c
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?

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?
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]);
}
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]);
}
```c
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]);
}
```
```c
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.

```c
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

```c
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

```c
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

| 'A' | 'l' | 'i' | 'c' | 'e' | '\0' | ... | 'B' | 'o' | 'b' | '\0' | ... | 'C' | 'a' | 't' | '\0' | ... |
|-----|-----|-----|-----|-----|-------|-----|-----|-----|-------|-----|-----|-----|-----|-------|-----|
| 2021 | 2022 | 2023 |
| 4.0 | 3.1 | 3.4 |
Q1 Discussion Block 2 of Worksheet
#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;
}
Abstraction

User / Programmer
Wants low complexity

Applications
Specific functionality

Software library
Reusable functionality

Operating system
Manage resources

Complex devices
Compute & I/O
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?

1 bit:
- 0
- 1

2 bits:
0 0
0 1
1 0
1 1

3 bits:
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
How many values?

1 bit:
- 0
- 1

2 bits:
- 0 0
- 0 1
- 1 0
- 1 1

3 bits:
- 0 0 0
- 0 0 1
- 0 1 0
- 0 1 1
- 1 0 0
- 1 0 1
- 1 1 0
- 1 1 1

4 bits:
- 0 0 0 0
- 0 0 0 1
- 0 0 1 0
- 0 0 1 1
- 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

16 values

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, unsigned long

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

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

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

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]

Digit #0: 1’s place, “least significant digit”

Digit #1: 10’s place

Digit #4: “most significant digit”

64025
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:

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

64025 =
6 \times 10^4 + 4 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 5 \times 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

Most significant bit \[10001111\] Least significant bit

\[
\begin{align*}
1 \times 2^7 &+ 0 \times 2^6 + \ldots + 1 \times 2^3 &+ 1 \times 2^2 &+ 1 \times 2^1 &+ 1 \times 2^0 \\
128 &+ &+ &8 &+ &4 &+ &2 &+ &1 \\
10001111 &\overset{\text{=}}{=} 143
\end{align*}
\]
Hexadecimal: Base 16

Indicated by prefixing number with \texttt{0x}

A number, written as the sequence of \( N \) digits,

\[ d_{n-1}...d_2d_1d_0, \]

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

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

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

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

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

A. 26  
B. 53  
C. 61  
D. 106  
E. 128
What is the value of 0x1B7 in decimal?

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

(Note: \(16^2 = 256\))

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

\[ i = 0 \]
\[ \text{while } (D > 0) \]
\[ \text{if } D \text{ is odd} \]
\[ \quad \text{set } b_i \text{ to 1} \]
\[ \text{if } D \text{ is even} \]
\[ \quad \text{set } b_i \text{ to 0} \]
\[ i++ \]
\[ D = D/2 \]

idea: example: D = 105 \quad b_0 = 1
Method 1: decimal value $D$, binary result $b$ ($b_i$ is $i$th digit):

\[
i = 0 \\
\text{while } (D > 0) \\
\quad \text{if } D \text{ is odd} \\
\quad \quad \text{set } b_i \text{ to } 1 \\
\quad \text{if } D \text{ is even} \\
\quad \quad \text{set } b_i \text{ to } 0 \\
\quad i++ \\
D = D/2
\]

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

Example: Converting 105
Method 1: decimal value $D$, binary result $b$ ($b_i$ is $i$th digit):

$$i = 0$$

while ($D > 0$)

  if $D$ is odd
    set $b_i$ to 1
  else if $D$ is even
    set $b_i$ to 0

  $i++$

$D = D/2$

idea:

<table>
<thead>
<tr>
<th>$D$</th>
<th>example: $D = 105$</th>
<th>$b_0$ = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D = D/2$</td>
<td>$D = 52$</td>
<td>$b_1$ = 0</td>
</tr>
<tr>
<td>$D = D/2$</td>
<td>$D = 26$</td>
<td>$b_2$ = 0</td>
</tr>
<tr>
<td>$D = D/2$</td>
<td>$D = 13$</td>
<td>$b_3$ = 1</td>
</tr>
<tr>
<td>$D = D/2$</td>
<td>$D = 6$</td>
<td>$b_4$ = 0</td>
</tr>
<tr>
<td>$D = D/2$</td>
<td>$D = 3$</td>
<td>$b_5$ = 1</td>
</tr>
<tr>
<td>$D = D/2$</td>
<td>$D = 1$</td>
<td>$b_6$ = 1</td>
</tr>
<tr>
<td>$D = 0$ (done)</td>
<td>$D = 0$</td>
<td>$b_7$ = 0</td>
</tr>
</tbody>
</table>

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)

\[
\begin{array}{cccccccc}
\frac{1}{d_6} & \frac{1}{d_5} & \frac{0}{d_4} & \frac{1}{d_3} & \frac{0}{d_2} & \frac{0}{d_1} & \frac{1}{d_0} \\
\end{array}
\]
What is the value of 357 in binary?

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?

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

\[
\begin{array}{cccccccc}
d_8 & d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1\\
\end{array}
\]

\[
2^0 = 1, \quad 2^1 = 2, \quad 2^2 = 4, \quad 2^3 = 8, \quad 2^4 = 16, \\
2^5 = 32, \quad 2^6 = 64, \quad 2^7 = 128, \quad 2^8 = 256
\]
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
 1010  6
+ 0100  + 4
 1010  10
```

Four bits give us range: 0 - 15
Unsigned Addition (4-bit)

• Addition works like grade school addition:

\[
\begin{array}{cccc}
\text{1} & 0110 & 6 & 1100 & 12 \\
+ & 0100 & + & 4 & + & 1010 & +10 \\
\hline
1010 & 10 & 1 & 0110 & 6 \\
\end{array}
\]

^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?

A: signed magnitude

B: Two’s complement

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)

• Borrow nice property from number line:

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

For an 8 bit range we can express 256 unique values:
• 128 non-negative values (0 to 127)
• 128 negative values (-1 to -128)
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 | exponent | fraction |
8 bits for exponent
23 bits for precision

\[ \text{value} = (-1)^{\text{sign}} \times 1.\text{fraction} \times 2^{(\text{exponent}-127)} \]

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 be 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
  - $[d_n \times 10^n] + [d_{n-1} \times 10^{n-1}] + \ldots + [d_2 \times 10^2] + [d_1 \times 10^1] + [d_0 \times 10^0]$
  - For any base system:
  - $[d_n \times b^n] + [d_{n-1} \times b^{n-1}] + \ldots + [d_2 \times b^2] + [d_1 \times b^1] + [d_0 \times b^0]$

• 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).