Number Representation

How do computers store numbers?

- All data in a computer is stored as binary using series of 1's and 0's
  - The way in which the bits are organized gives them meaning
- Each binary digit is called a **bit**
- In the C programming language, each variable has a **fixed number of bits** that is can use to represent different values

Decimal Representation

- People tend to do math and represent numerical values using a decimal representation
  - All numbers are made up of the digits 0-9

- How many numbers can we make with:
  - one decimal digit? 0 to 9 = 10 possible values (10^1)
  - two decimal digits? 0 to 99 = 100 possible values (10^2)
  - each time we add a digit we increase our value range by a power of 10

- n digits = 10^n possible values

Binary Representation

- All numbers are made up of the binary digits (bits) 0 or 1

- How many numbers can we make with:
  - one bit? 0, 1 = 2 possible values (2^1)
  - two bits? 0, 01, 10, 11 = 4 possible values (2^2)
  - each time we add a digit we increase our value range by a power of 2

- n digits = 2^n possible values

Storing Binary Data

- We group bits together in units of 8 called **bytes**.
- A byte is the smallest unit of data we can access from memory (RAM)
- **Data types** in C are used to represent values
  - Data types have a certain number of bits available for storage
  - The amount of storage is always in groups of 8 bits (1 byte)

Representing Decimal Values as Binary

- Remember that decimal is base 10 and binary is base 2
- In binary, each digit represents a power of 2 starting with 2^0

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	1	0	1	0

- When we translate from binary to decimal, we add up the powers of two positions that have a one
- 0 0 0 0 1 0 1 0 = $2^3 + 2^1 = 8 + 2 = 10$

How to convert decimal to Binary?

1. Divide the number by 2
2. Get the integer quotient for the next iteration
3. Get the remainder for the binary digit
4. Repeat the steps until the quotient is equal to 0

Convert 13 to binary => 1101

Divide by 2	Quotient	Remainder	Bit #
13 / 2	6	1	0
6 / 2	3	0	1
3 / 2	1	1	2
1 / 2	0	1	3

Int Data Type

- Stores positive and negative integer values
- According to the C standard must be **at least** 16 bits (2 bytes)
  - Most modern computers use 32 bits (4 bytes)
- Assuming we have 32 bits for an int, that is 2^{32} different numbers
  - Can be positive, negative, or zero
  - $2^{31} - 1$ positive numbers, 2^{31} negative numbers, and 0
    - Range: -2^{31} to $2^{31}-1$ (-2,147,483,648 to 2,147,483,647)
- A 32-bit unsigned int only stores positive numbers 0 to $2^{32} - 1$

Float Data Type

- Represents real numbers
  - Values with decimal precision
- Uses 32 bits of storage (4 bytes)
- Range is wider than an int
  - Max value is ~$3.4*10^{38}$
- While there are infinite values in the range only 2^{32} values can be represented
  - This leads to approximation

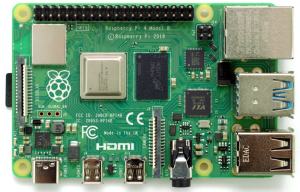
Double Data Type

- Like float and stores real numbers (decimal values)
- Unlike float, the storage is doubled to 64 bits (8 bytes)
- Max value is ~$1.8*10^{308}$
- Much greater precision
- Usually better to use double than float
  - Unless you are storing a very large number of decimal values or have limited resources

Data Types on Different Platforms

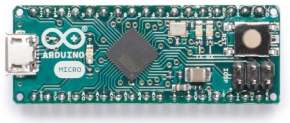
Mac Laptop (64-bit OS)

- _Bool: 8
- char: 8
- short int: 16
- int: 32
- **long int: 64**
- long long int: 64
- float: 32
- double: 64



Raspberry Pi (32-bit OS)

- _Bool: 8
- char: 8
- short int: 16
- int: 32
- long int: 32
- long long int: 64
- float: 32
- double: 64



Arduino (8-bit microcontroller)

- _Bool: 8
- char: 8
- short int: 16
- **int: 16**
- long int: 32
- long long int: 64
- float: 32
- **double: NA**