# Programming Paradigms

## CT331 Week 3 Lecture 1

Finlay Smith

finlay.smith@universityofgalway.ie

# C types

# Four basic arithmetic type specifiers

- int
- char
- float
- double

# Four type modifiers

- signed
- unsigned
- short
- long

Eg:

```
unsigned int number = 2;

Short int anotherNumber = 4;
```

# Integers

There is no fixed size for data types in C.

The size of an `int` *should* reflect the native word size of your system.

Eg.

A 32 bit system, `sizeof(int)` returns 4

A 16 bit system, `sizeof(int)`  returns 2

# Integer modifiers

**On most modern machines (32 and 64 bit):**

| | |
|---|---|
| sizeof(short int) | returns 2 (2 bytes, 16 bits) |
| sizeof(int) | returns 4 (32 bits) |
| sizeof(long int) | returns 8 (64 bits) |
| sizeof(long long int) | returns 8 (max size on most systems) |

# Integer modifiers

**Short int:**

```
00000000 00000000
…
11111111 11111111
```

0 to $2^{16}$ values (65536)

Not necessarily the numbers 0 to 65535

**Int:**

```
00000000 00000000 00000000 00000000
…
11111111 11111111 11111111 11111111
```

0 to $2^{32}$ values (4294967296)

# Integer modifiers

**unsigned short int:**

```
00000000 00000000 = 0
…
11111111 11111111 = 65535
```

Positive values only

**signed short int:**

```
10000000 00000000 = -32,768
…
00000000 00000000 = 0
…
01111111 11111111 = 32,767
```

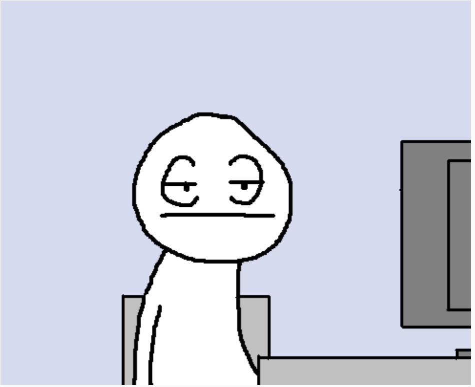
Positive and negative values…

Two’s complement!

# Two’s complement

1. Take the positive representation of the number (eg. 0011 = 3)
2. Flip the bits (0011 -> 1100)
3. Add 1 (1100 -> 1101)
4. *The two's complement of a number behaves like the negative of the original number in most arithmetic, and positive and negative numbers can coexist in a natural way.*

# Char



There is nothing special about the char data type in C.

A char is just a 1 byte integer.

# Char modifiers

sizeof(char) returns 1

Unsigned char => 00000000 - 11111111

0 255

signed char => 10000000 - 00000000 - 01111111

-128 0 127

# Printing letters

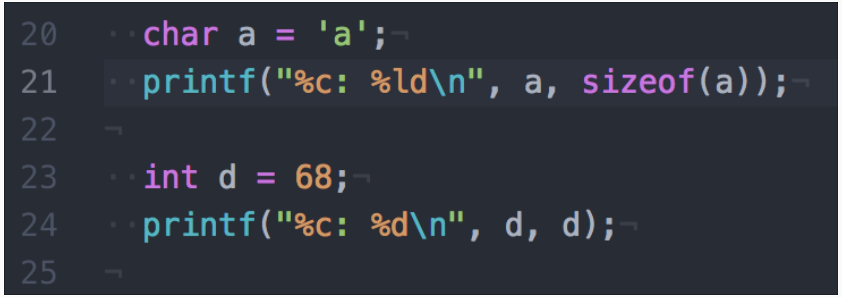
Q: If a char is just a regular integer, why does it print a letter?

A: Formatted output.

Q: What about using other integers?

A: ...

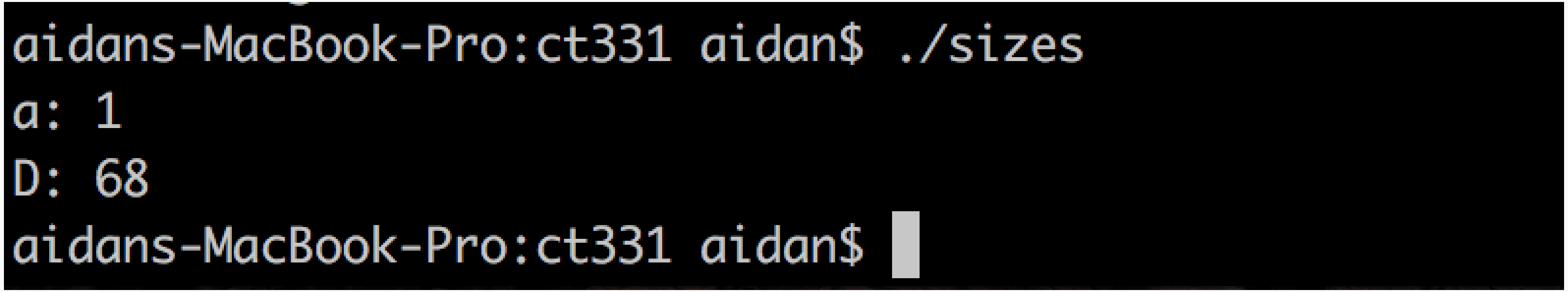
# Printing letters



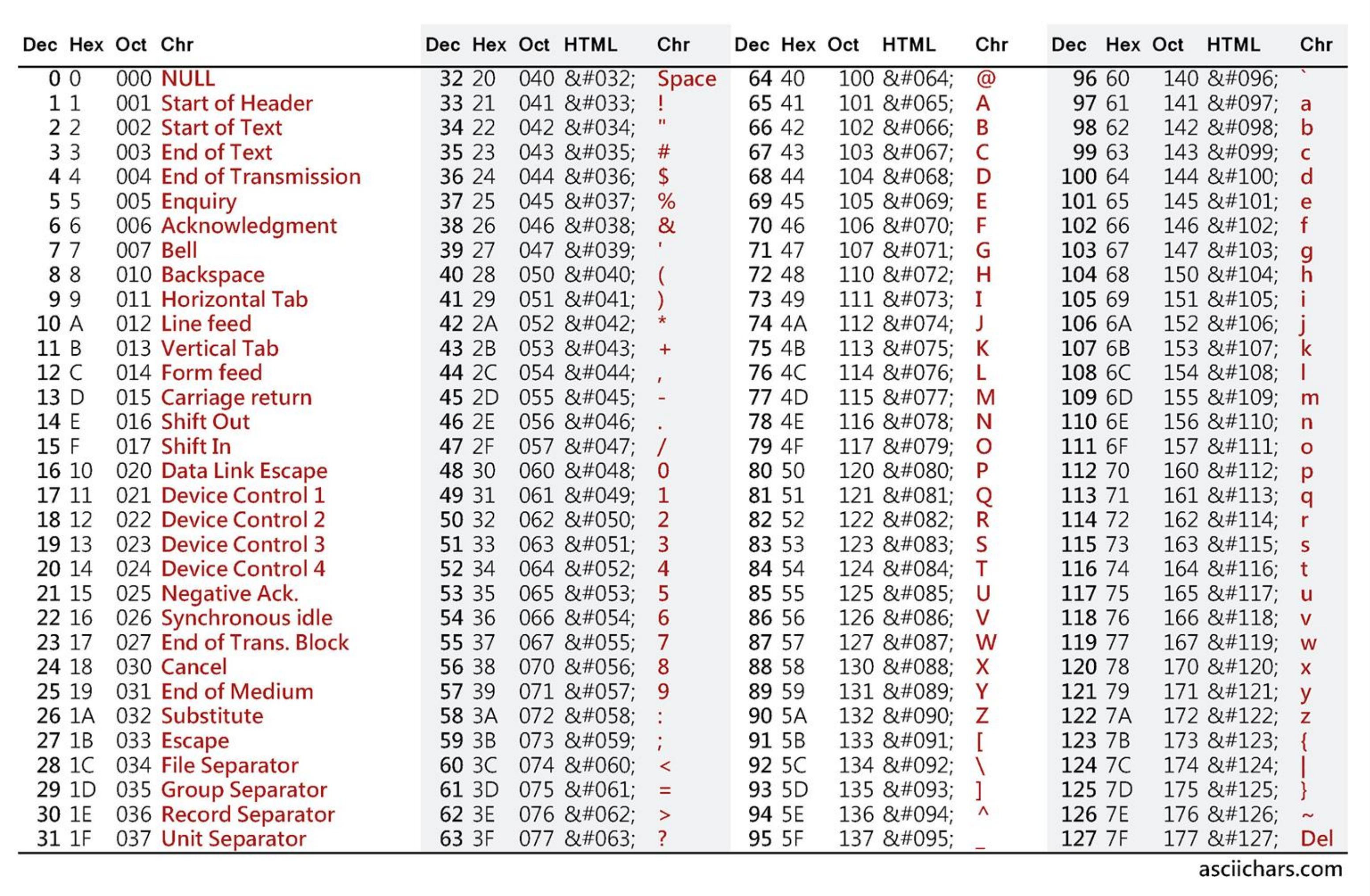
```c
  char a = 'a';
  printf("%c: %ld\n", a, sizeof(a));

  int d = 68;
  printf("%c: %d\n", d, d);

```



```
aidans-MacBook-Pro:ct331 aidan$ ./sizes
a: 1
D: 68
aidans-MacBook-Pro:ct331 aidan$ 
```



| Dec | Hex | Oct | Chr | Dec | Hex | Oct | HTML | Chr | Dec | Hex | Oct | HTML | Chr | Dec | Hex | Oct | HTML | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NULL | 32 | 20 | 040 | &#032; | Space | 64 | 40 | 100 | &#064; | @ | 96 | 60 | 140 | &#096; | ` |
| 1 | 1 | 001 | Start of Header | 33 | 21 | 041 | &#033; | ! | 65 | 41 | 101 | &#065; | A | 97 | 61 | 141 | &#097; | a |
| 2 | 2 | 002 | Start of Text | 34 | 22 | 042 | &#034; | " | 66 | 42 | 102 | &#066; | B | 98 | 62 | 142 | &#098; | b |
| 3 | 3 | 003 | End of Text | 35 | 23 | 043 | &#035; | # | 67 | 43 | 103 | &#067; | C | 99 | 63 | 143 | &#099; | c |
| 4 | 4 | 004 | End of Transmission | 36 | 24 | 044 | &#036; | $ | 68 | 44 | 104 | &#068; | D | 100 | 64 | 144 | &#100; | d |
| 5 | 5 | 005 | Enquiry | 37 | 25 | 045 | &#037; | % | 69 | 45 | 105 | &#069; | E | 101 | 65 | 145 | &#101; | e |
| 6 | 6 | 006 | Acknowledgment | 38 | 26 | 046 | &#038; | & | 70 | 46 | 106 | &#070; | F | 102 | 66 | 146 | &#102; | f |
| 7 | 7 | 007 | Bell | 39 | 27 | 047 | &#039; | ' | 71 | 47 | 107 | &#071; | G | 103 | 67 | 147 | &#103; | g |
| 8 | 8 | 010 | Backspace | 40 | 28 | 050 | &#040; | ( | 72 | 48 | 110 | &#072; | H | 104 | 68 | 150 | &#104; | h |
| 9 | 9 | 011 | Horizontal Tab | 41 | 29 | 051 | &#041; | ) | 73 | 49 | 111 | &#073; | I | 105 | 69 | 151 | &#105; | i |
| 10 | A | 012 | Line feed | 42 | 2A | 052 | &#042; | * | 74 | 4A | 112 | &#074; | J | 106 | 6A | 152 | &#106; | j |
| 11 | B | 013 | Vertical Tab | 43 | 2B | 053 | &#043; | + | 75 | 4B | 113 | &#075; | K | 107 | 6B | 153 | &#107; | k |
| 12 | C | 014 | Form feed | 44 | 2C | 054 | &#044; | , | 76 | 4C | 114 | &#076; | L | 108 | 6C | 154 | &#108; | l |
| 13 | D | 015 | Carriage return | 45 | 2D | 055 | &#045; | - | 77 | 4D | 115 | &#077; | M | 109 | 6D | 155 | &#109; | m |
| 14 | E | 016 | Shift Out | 46 | 2E | 056 | &#046; | . | 78 | 4E | 116 | &#078; | N | 110 | 6E | 156 | &#110; | n |
| 15 | F | 017 | Shift In | 47 | 2F | 057 | &#047; | / | 79 | 4F | 117 | &#079; | O | 111 | 6F | 157 | &#111; | o |
| 16 | 10 | 020 | Data Link Escape | 48 | 30 | 060 | &#048; | 0 | 80 | 50 | 120 | &#080; | P | 112 | 70 | 160 | &#112; | p |
| 17 | 11 | 021 | Device Control 1 | 49 | 31 | 061 | &#049; | 1 | 81 | 51 | 121 | &#081; | Q | 113 | 71 | 161 | &#113; | q |
| 18 | 12 | 022 | Device Control 2 | 50 | 32 | 062 | &#050; | 2 | 82 | 52 | 122 | &#082; | R | 114 | 72 | 162 | &#114; | r |
| 19 | 13 | 023 | Device Control 3 | 51 | 33 | 063 | &#051; | 3 | 83 | 53 | 123 | &#083; | S | 115 | 73 | 163 | &#115; | s |
| 20 | 14 | 024 | Device Control 4 | 52 | 34 | 064 | &#052; | 4 | 84 | 54 | 124 | &#084; | T | 116 | 74 | 164 | &#116; | t |
| 21 | 15 | 025 | Negative Ack. | 53 | 35 | 065 | &#053; | 5 | 85 | 55 | 125 | &#085; | U | 117 | 75 | 165 | &#117; | u |
| 22 | 16 | 026 | Synchronous idle | 54 | 36 | 066 | &#054; | 6 | 86 | 56 | 126 | &#086; | V | 118 | 76 | 166 | &#118; | v |
| 23 | 17 | 027 | End of Trans. Block | 55 | 37 | 067 | &#055; | 7 | 87 | 57 | 127 | &#087; | W | 119 | 77 | 167 | &#119; | w |
| 24 | 18 | 030 | Cancel | 56 | 38 | 070 | &#056; | 8 | 88 | 58 | 130 | &#088; | X | 120 | 78 | 170 | &#120; | x |
| 25 | 19 | 031 | End of Medium | 57 | 39 | 071 | &#057; | 9 | 89 | 59 | 131 | &#089; | Y | 121 | 79 | 171 | &#121; | y |
| 26 | 1A | 032 | Substitute | 58 | 3A | 072 | &#058; | : | 90 | 5A | 132 | &#090; | Z | 122 | 7A | 172 | &#122; | z |
| 27 | 1B | 033 | Escape | 59 | 3B | 073 | &#059; | ; | 91 | 5B | 133 | &#091; | [ | 123 | 7B | 173 | &#123; | { |
| 28 | 1C | 034 | File Separator | 60 | 3C | 074 | &#060; | < | 92 | 5C | 134 | &#092; | \ | 124 | 7C | 174 | &#124; | \| |
| 29 | 1D | 035 | Group Separator | 61 | 3D | 075 | &#061; | = | 93 | 5D | 135 | &#093; | ] | 125 | 7D | 175 | &#125; | } |
| 30 | 1E | 036 | Record Separator | 62 | 3E | 076 | &#062; | > | 94 | 5E | 136 | &#094; | ^ | 126 | 7E | 176 | &#126; | ~ |
| 31 | 1F | 037 | Unit Separator | 63 | 3F | 077 | &#063; | ? | 95 | 5F | 137 | &#095; | _ | 127 | 7F | 177 | &#127; | Del |

asciichars.com

# Floats and Doubles

- Floats and Doubles represent Real numbers in binary.
- There are an infinite number of real numbers in any given range.
- There are a finite number of binary representations given a finite number of bits

Therefore, any float or double is **necessarily imprecise.**

Ie. there is a lack of precision.

Ie. Floating numbers are not perfect.

# Floats and Doubles

$$m*b^e$$

m = Mantissa

b = base

E = exponent

http://www.cs.yale.edu/homes/aspnes/pinewiki/C(2f)FloatingPoint.html

# Floats and Doubles

**IEEE-754 floating point standard:**

```
bit  31 30    23 22                    0

     S  EEEEEEEE MMMMMMMMMMMMMMMMMMMMMMM
```

(`Double` uses 64 bits  - 11 exponent bits, 52 mantissa bits)