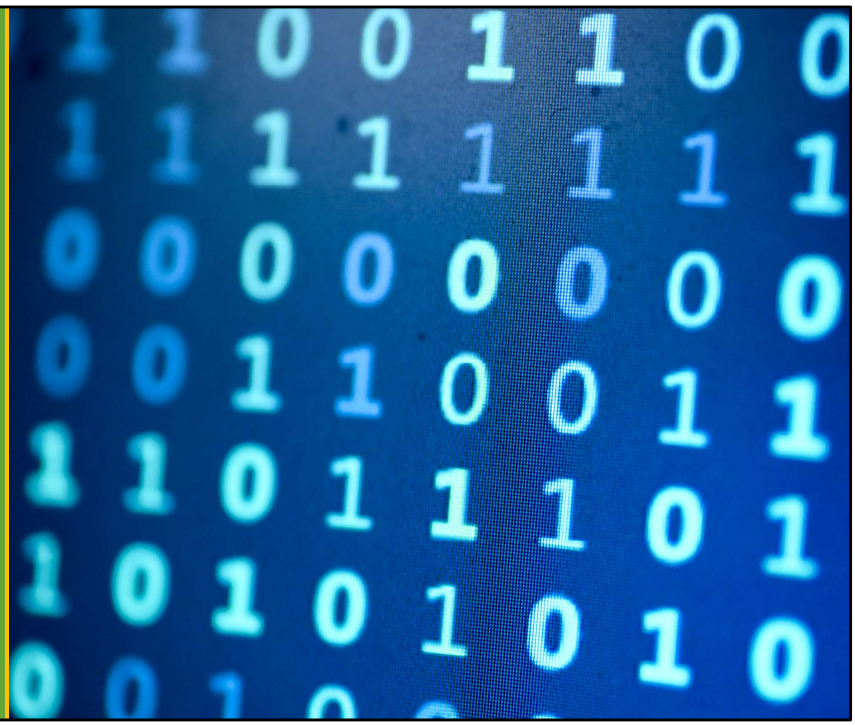
CS221
C and Systems Programming



Bit operations in C

1	1	1	0	0	0	0	1

```
char val1 = -31;
```

0	0	0	0	0	1	1	1

```
char val2 = 7;
```

Bit operations in C - and

1	1	1	0	0	0	0	1

```
char val1 = -31;
```

0	0	0	0	0	1	1	1

```
char val2 = 7;
```

0	0	0	0	0	0	0	1

```
val1 & val2;
```


Show "truth tables" for AND, OR, etc.

Bit operations in C - or

1	1	1	0	0	0	0	1

```
char val1 = -31;
```

0	0	0	0	0	1	1	1

```
char val2 = 7;
```

1	1	1	0	0	1	1	1

```
val1 | val2;
```

Bit operations in C - xor

1	1	1	0	0	0	0	1

```
char val1 = -31;
```

0	0	0	0	0	1	1	1

```
char val2 = 7;
```

1	1	1	0	0	1	1	0

```
val1 ^ val2;
```


Show "truth tables" for AND, OR, XOR

Bit operations in C - not

```
char val = -31;
```

1	1	1	0	0	0	0	1

```
~val;
```

0	0	0	1	1	1	1	0

Bit operations in C – left-shift

```
char val = -31;
```

1	1	1	0	0	0	0	1

```
val << 2;
```

1	0	0	0	0	1	0	0

Bit operations in C – right-shift

```
char val = -31;
```

1	1	1	0	0	0	0	1

```
unsigned char val2 = 129;
```

1	0	0	0	0	0	0	1

```
val >> 2;
```

1	1	1	1	1	1	0	0

```
val2 >> 2;
```

0	0	1	0	0	0	0	0

Sign-extension for signed values. NOT for unsigned values. Try in C!

Ugh! Why do we care?

- Hardware parts of computers will often set/get bit values in software registers to communicate…
- $x2^N$ → <<N
- $\div 2^N$ → >>N
- Some specialized algorithms (encryption, compression, random #s) manipulate bits

```
const unsigned NumBits = 5;
unsigned counter = 0;
while (1) {
 counter = (counter+1) & ( (1<<NumBits) - 1);
}
```


Show NumBits example: a "wraparound counter" without an if() statement
- compiler smart enough to precompute the constant value

See makeRand.c

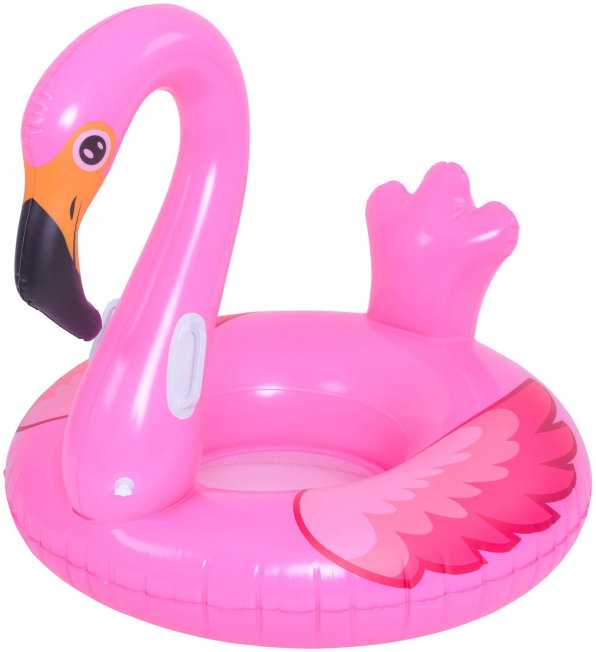
Some practice with `char` and `unsigned char`

```
11 << 4
11 & 12
15 | 16
-128 >> 7
-128 >> 6
-128 << 6
96 & 48
96 | 48
```

```
1 & -1
1 ^ -1
-16 ^ 15
(3 << 1) & 3
47 & 47
-99 | -99
```


first one: char or unsigned char?

Floats and doubles



How are `float` and `double` represented?

Sign bit

"Mantissa"

Exponent

$$\text{Value} = (-1)^{\text{SignBit}} \times \text{Mantissa} \times 2^{\text{Exponent}}$$

How are `float` and `double` represented?

Examples

Sign Bit	Mantissa	Exponent	Value
1	1	0	-1.0
0	5	-2	1.25
0	3	-1	1.5
1	127	3	-1016.0

How are `float` and `double` represented?

Some details

- 2 ^ (Exponent – K), so Exponent can be unsigned
- Value "0" represented with Exponent == 0
- Special values **+∞**, **- ∞**, **NaN** represented with all-1's Exponent

"Normalization"

- Shift Mantissa to the left and subtract 1 from Exponent until MSB of Mantissa is '1'
- Maximizes # of significant digits, maximizes numerical accuracy

`float` and `double`

float
- 1 sign bit
- 8 exponent bits
- 24 mantissa bits

double
- 1 sign bit
- 11 exponent bits
- 53 mantissa bits

What's strange about these formats?

MS bit of Mantissa always '1' so we don't need to store it

New C syntax

Two new data types

```
long long bigInteger;

long double bigFloating;
```


How big are these?
- 8, 16 on my computer

There is unsigned long long also.

These give a lot more numerical accuracy for when you need it. But calculations are slower.
SEE: machineeps.c