Representing Data with Bits

bits, bytes, numbers, and notation

Show powers, strategies.

ex

conversion and arithmetic

$19_{10} = ?_2$

$1001_2 = ?_{10}$

$240_{10} = ?_2$

$11010011_2 = ?_{10}$

$101_2 + 1011_2 = ?_2$

$1001011_2 \times 2_{10} = ?_2$

ex

bitwise operators

Bitwise operators on fixed-width **bit vectors**.

AND & OR | XOR ^ NOT ~

```
  01101001        01101001        01101001
& 01010101      | 01010101      ^ 01010101      ~ 01010101
  --------      --------      --------      ----------
  01000001

                                              01010101
                                            ^ 01010101
                                            ----------
```

Laws of Boolean algebra apply bitwise.

e.g., DeMorgan's Law: ~(A | B) = ~A & ~B

ex

bitwise operators in C

`& | ^ ~` apply to any *integral* data type

`long, int, short, char, unsigned`

Examples (`char`)

```
~0x41 =

~0x00 =

0x69 & 0x55 =

0x69 | 0x55 =
```

Many bit-twiddling puzzles in upcoming assignment

logical operations in C

ex

&& || ! apply to any "integral" data type
long, int, short, char, unsigned

0 is **false** **nonzero** is **true** **result** always **0 or 1**

early termination a.k.a. **short-circuit evaluation**

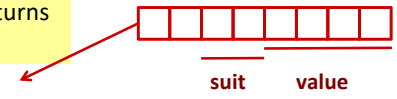
Examples (**char**)

```
 !0x41 =
 !0x00 =
!!0x41 =

0x69 && 0x55 =
0x69 || 0x55 =
```


Compare Card Values

ex

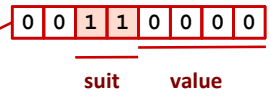
mask: a bit vector that, when bitwise ANDed with another bit vector *v*, turns all *but* the bits of interest in *v* to 0




```
#define VALUE_MASK

int greaterValue(char card1, char card2) {




}

 char hand[5];       // represents a 5-card hand
 char card1, card2;  // two cards to compare
 ...
 if ( greaterValue(hand[0], hand[1]) ) { ... }
```


Compare Card Suits

mask: a bit vector that, when bitwise ANDed with another bit vector *v*, turns all *but* the bits of interest in *v* to 0




```
#define SUIT_MASK 0x30

int sameSuit(char card1, char card2) {
  return !((card1 & SUIT_MASK) ^ (card2 & SUIT_MASK));

  //same as (card1 & SUIT_MASK) == (card2 & SUIT_MASK);
}
```

```
char hand[5];       // represents a 5-card hand
char card1, card2;  // two cards to compare
...
if ( sameSuit(hand[0], hand[1]) ) { ... }
```


Shift and Mask: extract a bit field

ex

Write C code:

extract *2nd most significant byte* from a 32-bit integer.

given x = 01100001 01100010 01100011 01100100

should return: 00000000 00000000 00000000 01100010

All other bits are zero.

Desired bits in least significant byte.