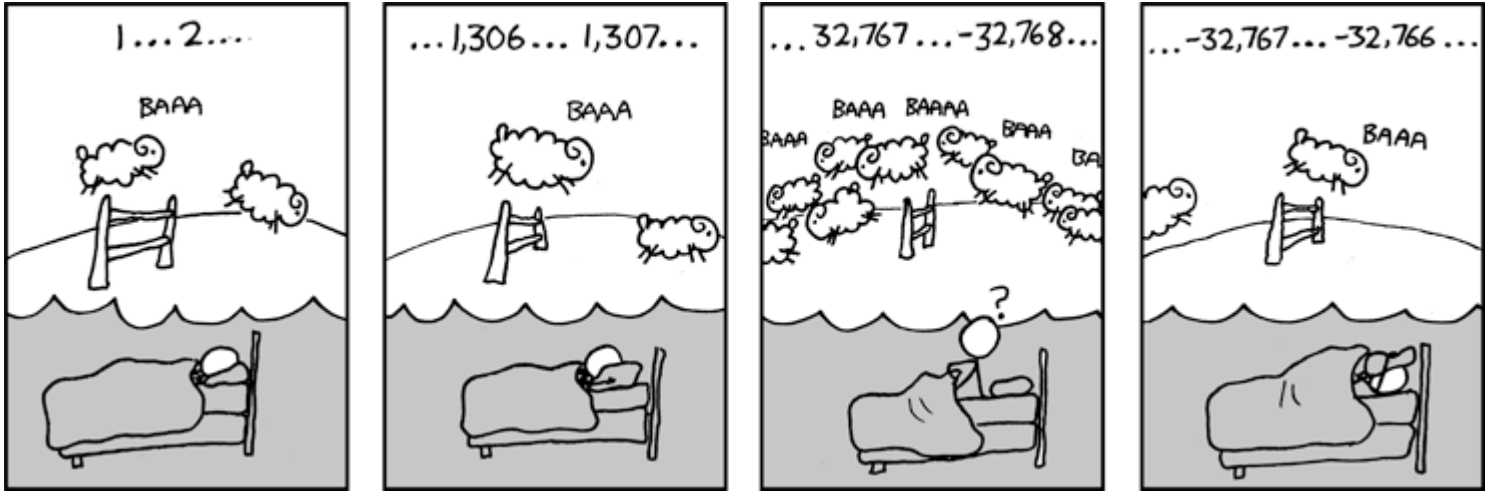
CSCI 2330


1...2...
BAAA
...1,306... 1,307...
BAAA
...32,767...-32,768...
BAAA BAAAA
BAAA
BAAA
?
...-32,767... -32,766...
BAAA

TODO

- Assignment 0 is assigned (due September 11)
- Assignment 1 is assigned (due September 24)

Overview for today

- Recap sign extension
- Conclusion of integer representation
- Lab 1

Integer Logic Exercises

Promotion hierarchy:

1) unsigned long
2) long
3) unsigned int
4) int
5) Everything narrower undergoes int promotion

```
unsigned int ux; int x; int y;
```

- `ux >= 0` always true?

```
unsigned int ux; int x; int y;
```

- `ux >= -1` always true?

`unsigned int ux; int x; int y;`

- If `x > 0 && y > 0`

  `(x + y) > 0` always true?

`unsigned int ux; int x; int y;`

- If `x >= 0`

  `-x <= 0` always true?

`unsigned int ux; int x; int y;`

- If `x <= 0`

  `-x >= 0` always true?

`unsigned int ux; int x; int y;`

- If `x > y`

  `-x < -y` always true?

`unsigned int ux; int x; int y;`

- If `x & 0x7 == 0x7`

  `(x << 30) < 0` always true?

```
unsigned int ux; int x; int y;
```

- `(x | -x) >> 31 == -1` always true?

Widening a type

- How do we determine the new bits?
- We determine them to preserve its value
- If the original type was unsigned, *zero extend*
- If the original type was signed, *sign extend*

What do shifts do arithmetically?

- $x << n == x * 2^n$
- $x >> n == x / 2^n$ (floor division)

Why sign-extend signed right shifts?

- To preserve this arithmetic property for right bitshifts of signed numbers:
- $x >> n == x / 2^n$ (floor division)
- We call a sign-extended right bitshift an *arithmetic shift*.

(Assume shifted values are signed)

- 8 >> 2 → ?

(Assume shifted values are signed)

- 8 >> 2 → 2

(Assume shifted values are signed)

- -8 >> 2 → ?

(Assume shifted values are signed)

- -8 >> 2 → -2

(Assume shifted values are signed)

- 1 >> 2 → ?

(Assume shifted values are signed)

- 1 >> 2 → 0

(Assume shifted values are signed)

- -1 >> 2 → ?

(Assume shifted values are signed)

- -1 >> 2 → -1

- Recall: floor(-0.25) = -1
- Therefore, -1 / 4 == -1

(Assume shifted values are signed)

- (5 == 5) << 1 → ?

(Assume shifted values are signed)

- (5 == 5) << 1 → 2

By the way…

- Fill in the logical operator: “x and y”
- x ?? y

By the way…

- Fill in the logical operator: “x and y”
- x && y

By the way…

- Fill in the logical operator: “x or y”
- x ?? y

By the way…

- Fill in the logical operator: “x or y”
- x || y

By the way…

- Fill in the logical operator: “x but y”
- x ?? y

By the way…

- Fill in the logical operator: “x but y”
- x && y

How to prevent overflow/underflow

- ???

How to prevent overflow/underflow

- signed → unsigned (maybe)
- int → long
- long → __int128 (!?)
- “BigInt” / GMP

Python: all integers are infinite precision

```
>>> 123 ** 456
992500687720988567008314620574696326372959408198869005198162988813828671047493990779211286614261446380554242369362718724928003527416499021181438196726015699981001207904967595176364654458956257416098662099005001984071532446047789680169630280503102614176159144687299182406854878786176459769390634643579861657117309763994785076492286863414669671679101266533421349427448514638999274870924866109771461127635671016726459531321964814393398730170881404146612711985003332557130961423351514146306516830655187840812036784877030028020820912366035190262568806244996817813872275740354848312715156831237421490955692604636096559777009388445806119312464951662086955403136981400116380273225662526897808381363518287953142721621112222311709017156123557013475523715300136938553798348656670600146433024591004297836539669137830022907842834556282833554705299329560514844771293338811599302127586876027950885792304316616960102321873904366016141456032419023866634425201607355665 61
```

```
>>> math.log2(123 ** 456)
3165.7866144346935
```

Real world examples of overflow

- These bugs have had a surprisingly large impact on human history

Y2K bug

- “19” + YY
- 1998
- 1999
- ~~2000~~ 1900

Year 2038 problem

- How else can computers represent time?
- Unix timestamp: number of seconds since Unix epoch (January 1, 1970)
- On 32-bit computers, represented as a 32-bit signed integer
- How high can that get until it overflows?

```
>>> import datetime
>>> datetime.datetime(1970, 1, 1) + datetime.timedelta(seconds=0x7fffffff)
datetime.datetime(2038, 1, 19, 3, 14, 7)
```

Fixes

- Switch to unsigned would buy a little more time
- Decision: switch to 64-bit signed integer
  - Will all software get patched? Can all software be patched?
- 64-bit machines always used 64-bit time

When to use signed vs. unsigned?

- Unsigned is more predictable across machines
- Unsigned can predictably wrap around
- Signed is less predictable
- Some machines aren’t even two’s complement

When to use signed vs. unsigned?

- **Never** negative → unsigned
- Can be negative and will **never** overflow/underflow → signed
- Any other combination → you need something higher level (BigInt, another language, etc.)

Print letters forwards…

```
int main(void) {
    char *string = "hello";
    for (unsigned int i = 0; i < strlen(string); ++i) {
        printf("%c\n", string[i]);
    }
    return 0;
}
```

Print letters in reverse…!?

```
int main(void) {
    char *string = "hello";
    for (unsigned int i = strlen(string) - 1; i >= 0; --i) {
        printf("%c\n", string[i]);
    }
    return 0;
}
```

Print letters in reverse…

```
int main(void) {
    char *string = "hello";
    for (unsigned int i = strlen(string); i-- > 0; ) {
        printf("%c\n", string[i]);
    }
    return 0;
}
```

Digital computers

- In electronic computers, has to do with voltage being either on or off
- Therefore, only 0s or 1s

Digitization

- Record in 0s and 1s
- [0, 1, 0, 1, 0, 1]
- → [0.03, 0.94, 0.02, 0.99, 0.04, 0.95]
- → [0, 1, 0, 1, 0, 1]
- → [0.02, 0.95, 0.01, 0.98, 0.03, 0.96]
- → [0, 1, 0, 1, 0, 1]

Digital is not necessarily base 2

- In fact, *digit* suggests base 10

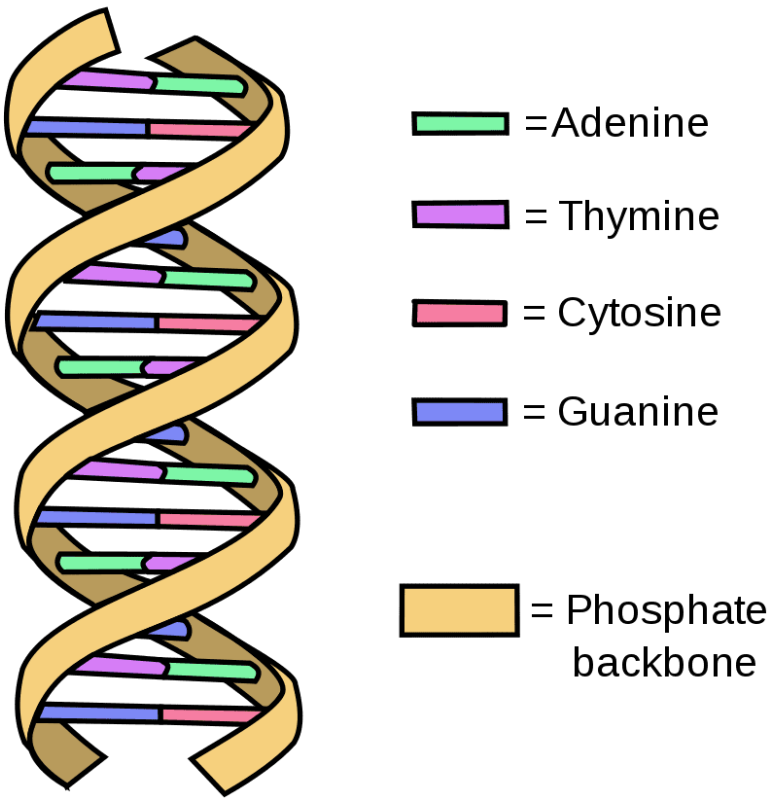
Non-binary Computers

- Soviets: “Setun” computer (1950)
- *Ternary*
- -1, 0, 1 (“trit”)
- Hardware challenges
- Social challenges

Base 4?

DNA sequences

- Bases: A, T, C, G
- Remember: digital signals are easier to copy without errors
- Outside of DNA sequences, we are largely analog



DNA

Lab 1

- Lab 1 is assigned
- Due September 24
- Start work early
- This one is tougher than it looks!

Work in groups!

- Not *required*
- But you might enjoy it!
- See Syllabus for rules (nothing complicated)

Lab 1

- Superficially, you’re filling in C functions
- But in reality you’re solving tricky bit puzzles
- And you don’t need to know much C at all

TODO

- Assignment 0 is assigned (due September 11)
- Assignment 1 is assigned (due September 24)