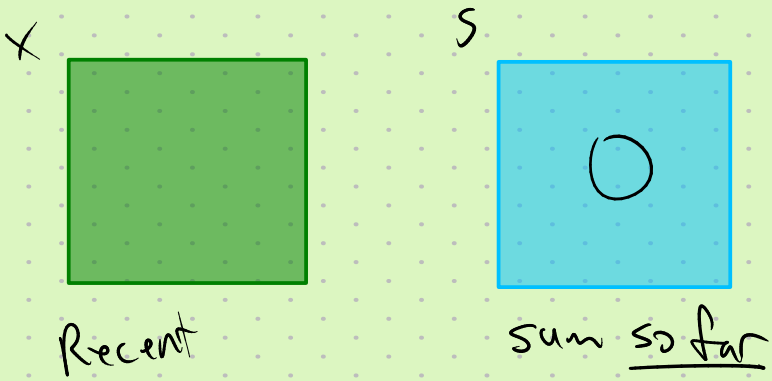
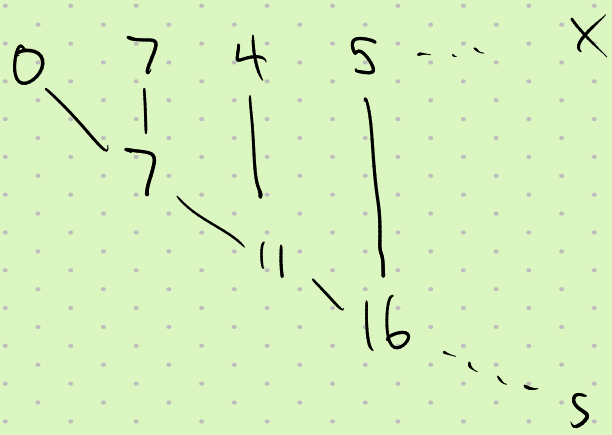
Last time: computing sum of integers from stdin.

Idea: 2 variables:



Then we kind of fold each new value into a running total



Note: this is a special case of the "fold" pattern. Say you have a binary operator [?] and you want to compute

$$x_1 \boxed{?} x_2 \boxed{?} x_3 \boxed{?} \cdots \boxed{?} x_n$$

([?] could be $+$, $*$, ...)

Suppose also, you have a "neutral element" e:

for all x, $x \boxed{?} e = e \boxed{?} x = x$.

E.g. $e = 0$ for $+$, $e = 1$ for $*$.

Then you have the following meta solution:

```
s = e;
while(cin >> x)
    s = s [?] x;
```

Note: our largest # program follows this exactly!

$a \boxed{?} b \overset{\text{def}}{=} \max(a,b)$ ($\overset{\text{def}}{=}$ signals a new definition)

$e = -\infty$ ($\approx$ INT_MIN)

Similarly, if $\boxed{?} = \min$, $e = \infty$ ($\approx$ INT_MAX),

meta program would compute smallest integer..

There are tons of examples... Maybe read about functional programming.

Review of basic ingredients / tools.

- allocate storage / memory, e.g.

```
int x;
float y;
char c;
string s;
bool b;
```

- Overwrite stuff, e.g.

```
x = 7;
y = 2.0;
c = 'A';
s = "abcdef";
b = (x == 99);
```

- Do something maybe (≤ 1 times), e.g.

```
if (x < 100) {
    // thing to do if x < 100
    ⋮
}
```

See also else, else if ...

- Do something repeatedly (until a condition is broken)