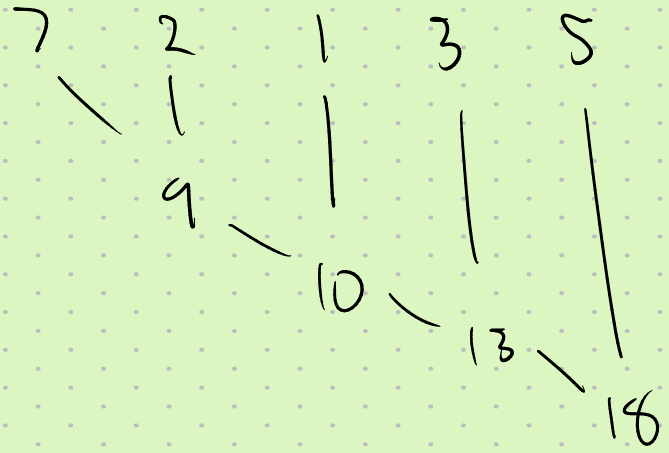
Last time: computing a sum of many integers:

Idea: keep track of a "sum so far", and keep adding new values to it.



More generally, suppose you have a binary operator $\boxed{?}$ & list of values, and want to compute

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

Suppose also, $\exists$ a value e that is "neutral" for $\boxed{?}$.

"there exists"

I.e., $\forall x,\ x \boxed{?} e = e \boxed{?} x = x$

"for all"

for $\boxed{?} = +$, $e = 0$

for $\boxed{?} = *$, $e = 1$

Then we have the following meta-solution:

```
s = e; // set s to neutral element
while(cin >> x){
    s = s [?] x;
}
cout << s << "\n";
```

Other examples:

- computing a product. $[?] = *$, $e = 1$.
- computing the max. $x [?] y = \max(x, y)$, $e = -\infty$ ($\approx$ INT_MIN)
- computing the min: $x [?] y = \min(x, y)$, $e = \infty$ ($\approx$ INT_MAX)

Even project 2 could even be seen as an example!

Note: this is called a "fold" in functional programming.

Review...

Building blocks / tools we have so far.

- variables.
  Allocating memory / new post it note

```
int  x;
char c;
bool b;
...
```

- assignment statements.
  How to write to a variable.

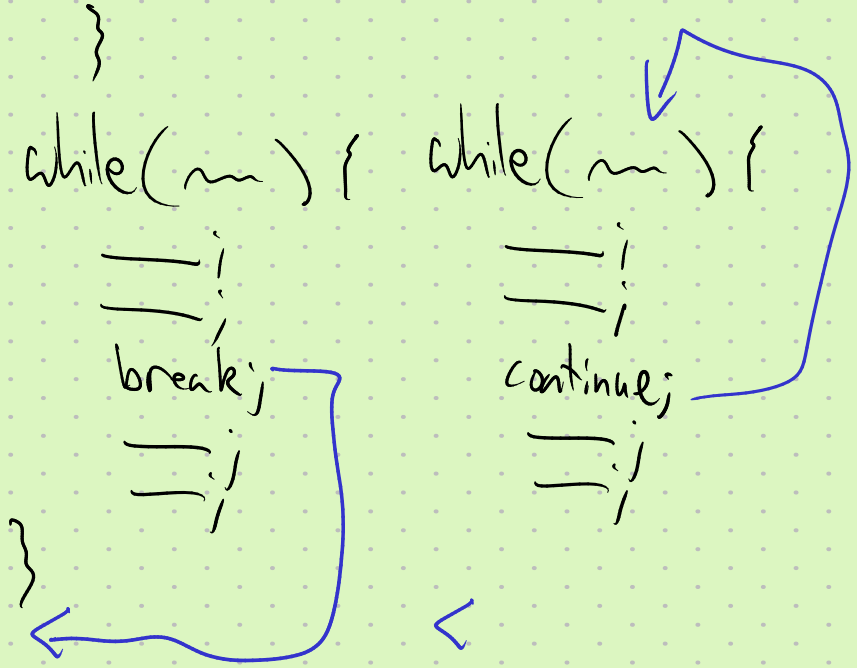
```
x = 73;
c = 'A';
b = true;
```

- conditionals (if/else)

```
if(<boolean expr>){
    statements...
    ...
}else if(<boolean expr>)
    other statements...
    ...
}else if(<boolean expr.>){
    ...
}else{
    statements...
    ...
}
```

- loops (while/for)
(also remember that "break;" is a thing, as is "continue;")

```
while(<boolean expr.>){
    — statements ...
    — — —
    — — —
}
```



(Note: break & continue will usually appear in an if statement. See our first program for an exaple.

for loops:

```
for(<setup statement>; <boolean expr.>; <update stat>){
    statements ...
    :
} // sequence: 1, 2, 3, 4
```

1 2 4

until (2) is false