EVERY ALGORITHM MUST HAVE:

① description of valid inputs

② description of valid outputs, in terms of the inputs

VARIABLES

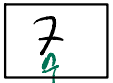
"assignment": sets the value to be 7

$x \leftarrow 7$

can just write this to introduce a new variable x and assign its value

"a box"

x ← label for the box



value stored inside the box (cannot be empty!)

ARRAYS

$A[1, \ldots, n]$

first index is 1!

1st thing in the array A = A[1]

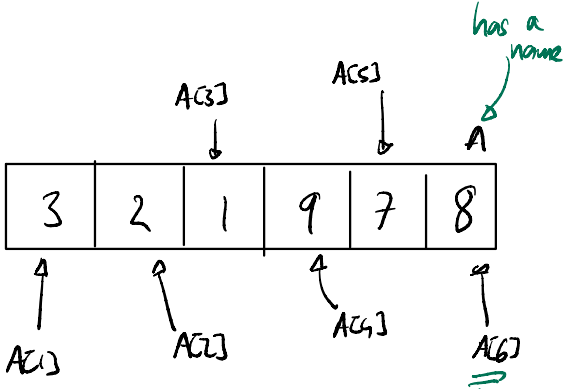
2nd thing in the array A = A[2]

⋮

$A[1] \leftarrow 7$
$A[2] \leftarrow A[1]$

] can read from and assign to elements of the array

"a collection of boxes"



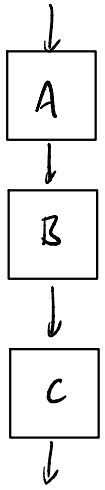
length is fixed and known in advance

DIRECT SEQUENCING

to do A, then B, then C, just write them one after another:

```
A
B
C
```

```
A
B   =>
C
```



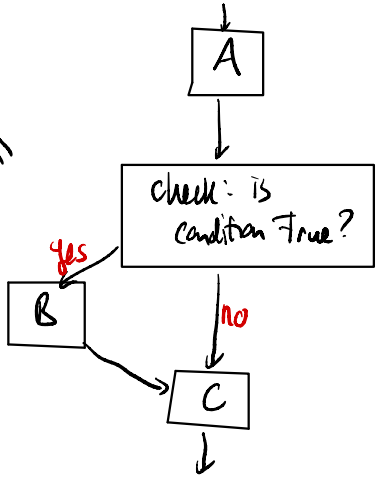
CONDITIONAL SEQUENCING

First construct:

```
if (condition) then
    (something)      // only happens if condition is true
```

```
A
if (condition) then
    B
C
```

C is not indented, so happens no matter what



Second construct:

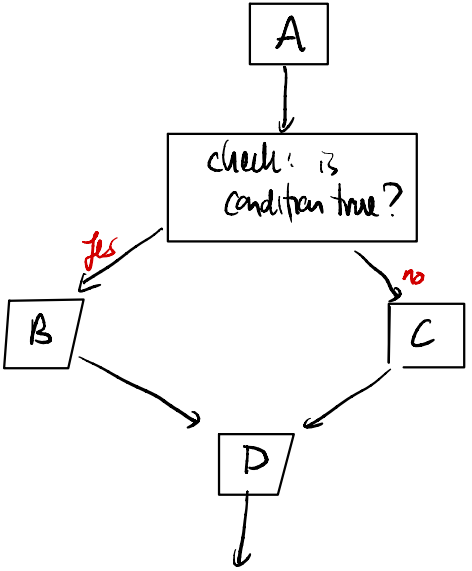
```
if (condition) then
    (something)
else
    (something else)
```

happens if condition is true

happens if condition is false

```
A
if (condition) then
    B
else
    C
D
```

D is not indented, happens no matter what



BOUNDED ITERATION

A way to do something a fixed number of times:

for i ← (start) to (finish) do
(something)

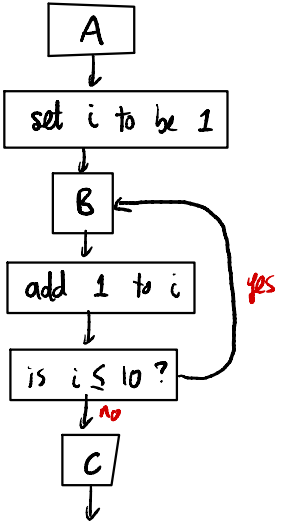
- an iterator: given to you by the algorithm. You can't assign values to it!
- numbers (can be something like n, or a variable)
- can depend on i (eg: sum ← sum + i)

⚠ If you know it will take a fixed number of steps, you MUST use a for-loop.

"do this 10 times" (like: break 3 eggs)

```
A
for i ← 1 to 10 do
    B
C
```

=



EXERCISE: what does the picture look like for:

```
sum ← 0
for i ← 1 to 10 do
    sum ← sum + i
```

OUR FIRST ALGORITHM

Input: positive number n

Output: sum of first n positive numbers

$\text{SUM} \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**

$\quad \text{SUM} \leftarrow \text{SUM} + i$

return SUM

Conditional iteration

Perform *something* repeatedly so long as some *condition* remains true

General format:

while *condition* **do**
| *something*

Conditional iteration

Perform *something* repeatedly so long as some *condition* remains true

General format:

while *condition* **do**
  something

Explanation:

- Executing *something* can affect the value of *condition*
- If *condition* is false *something* is not executed
- If *condition* is true *something* is executed and . . . evaluation repeats itself
- The value of *condition* may remain true forever

Conditional iteration: Example

- Summing numbers:

Input: positive number n
Output: sum of first n positive numbers
$\text{SUM} \leftarrow 0$
$\text{ITER} \leftarrow 1$
while $\text{ITER} \leq n$ **do**
    $\text{SUM} \leftarrow \text{SUM} + \text{ITER}$
    $\text{ITER} \leftarrow \text{ITER} + 1$
return SUM

CONDITIONAL ITERATION

Keep doing something while the condition remains true:

```
while (condition) do
    (something)
```

⚠ if the condition is always true, this will run forever!

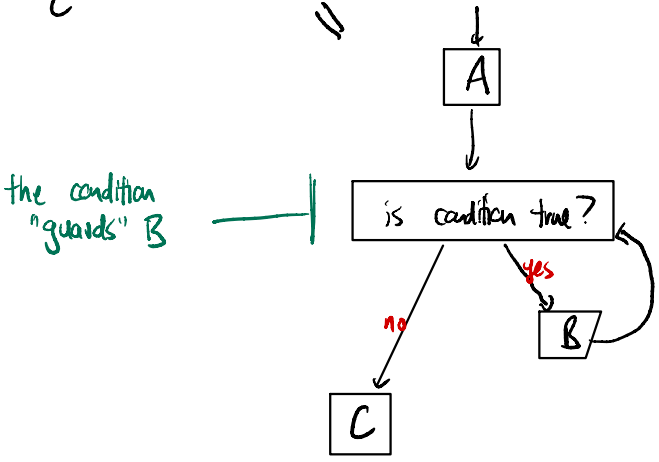
⚠ if the condition is never true, (something) will not run!

Unlike with for-loops, you handle the condition. So you must be sure why the loop will terminate when you write it!

"do this until the condition is false"

(like: beat the eggs until they are fluffy)

```
A
while (condition) do
    B
C
```

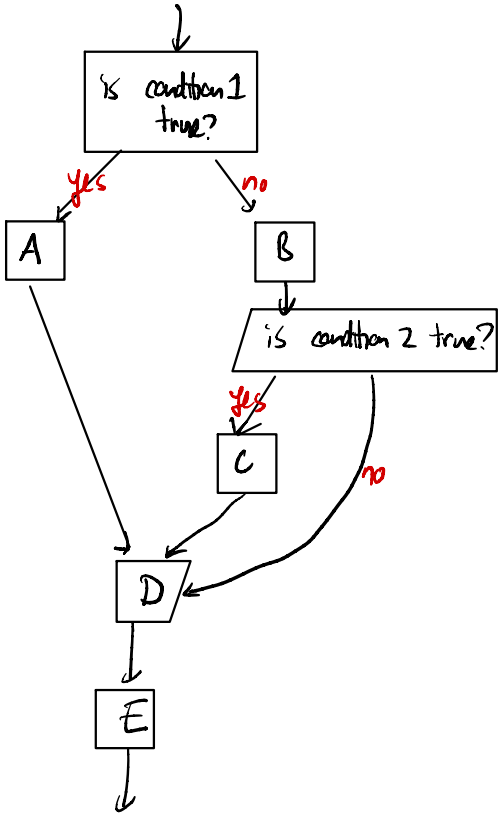


the condition "guards" B

INDENTATION MATTERS

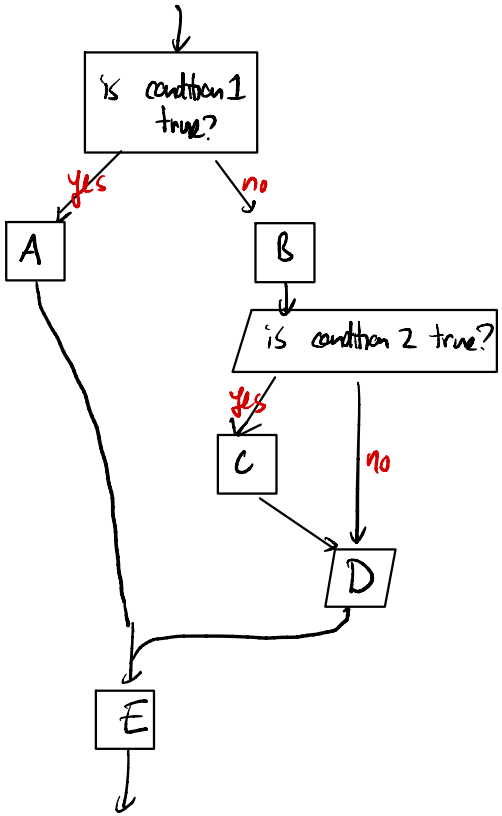
```
if (condition 1) then
    A
else
    B
    if (condition 2) then
        C
D
E
```

=



```
if (condition 1) then
    A
else
    B
    if (condition 2) then
        C
    D
E
```

=



EXERCISE: practice this for yourself on for-loops and while-loops. Can you draw any of the algorithms we study as a picture?