Example: Fibonacci sequence.

$a_0 \quad a_1 \quad a_2 \quad \cdots$

$1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8$

$$a_n = \begin{cases} 1 & \text{if } n < 2 \\ a_{n-1} + a_{n-2} & \text{else} \end{cases}$$

input: $n \in \mathbb{Z}$, $n \geq 0$.

output: a_n, the n^{th} term of the sequence.

$1 \quad 1 \quad 2 \quad \cdots \; \_\_ \quad \_\_$

Issue: we have bounded space to work with (limited # of variables)

Good news: only a few #'s are relevant at any given point.

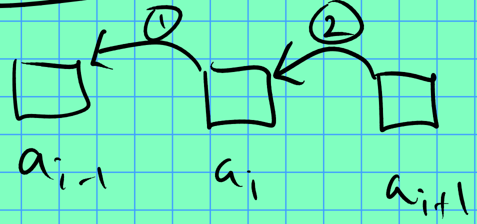
a_i, a_{i-1}, i. This enables us to get

$$a_{i+1} = a_i + a_{i-1}$$

$$\left.\begin{array}{l} a_i \equiv \text{most recently computed term} \\ a_{i-1} \equiv \text{term before } a_i \\ i \equiv \text{\# terms computed} \end{array}\right\} \text{invariant}$$

How to move one step while preserving the invariant?

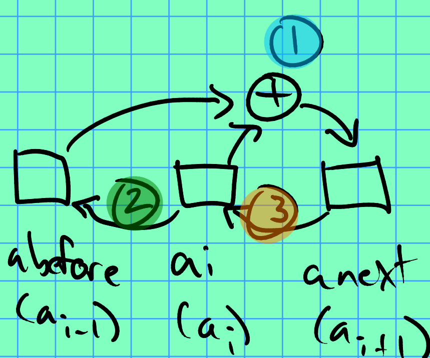
$i \to i+1$



$a_{i-1} = a_i;$
$a_i = a_{i+1};$

Now in C++:

```
int n;                    // a0 a1 ...
                          //  1  1  2  3  5 ...
cin >> n;
int i = 1;
int ai, abefore;  // ai ≡ a_i , abefore ≡ a_{i-1}
ai = 1;
abefore = 1;
// Note values of ai, abefore, i are
// consistent with the meaning we save them!
// I.e., the invariant is established.
if (n < 2) {
    cout << 1;
    return 0;
}
// else n ≥ 2
while (i < n) {
    i++;
    int anext = abefore + ai;  // 1
    abefore = ai;  // 2
    ai = anext;  // 3
```



```
}
// here, we know i==n & ai = aj = an.
cout << ai ;
```