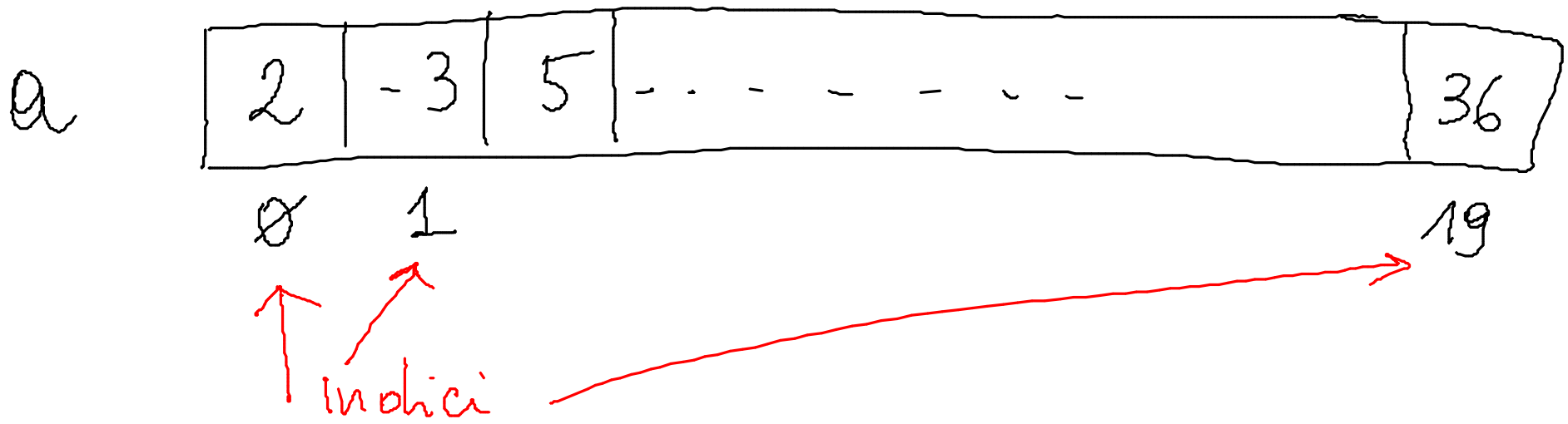
ARRAY in C

Sequenza di VARIABILI di tipo omogeneo.

```
int a[20];
```

- `int`: tipo dei valori
- `a`: nome dell'array
- `20`: lunghezza (il numero di elementi)




```
int x;
```

a[0] ← primo elemento dell'array

a[1] ← secondo " "

a[2]

⋮

a[n-1] ← l'ultimo elemento dell'array

```
a[0] = a[0] + 1;
```

Variabile destinazione dell'assegnamento

punto di un'espressione

~~int n;~~
~~int a[n];~~

NO

```
{ int a[10]; int x = 10;
    ..... legge i valori dell'array
  int i = 0; int trovato = 0;
  while ( i < 10 && !trovato )
    if (a[i] == x)        trovato = 1;
                  else    i = i+1;
```

→ il valore di trovato è TRUE se $x \in a$
è FALSE se $x \notin a$

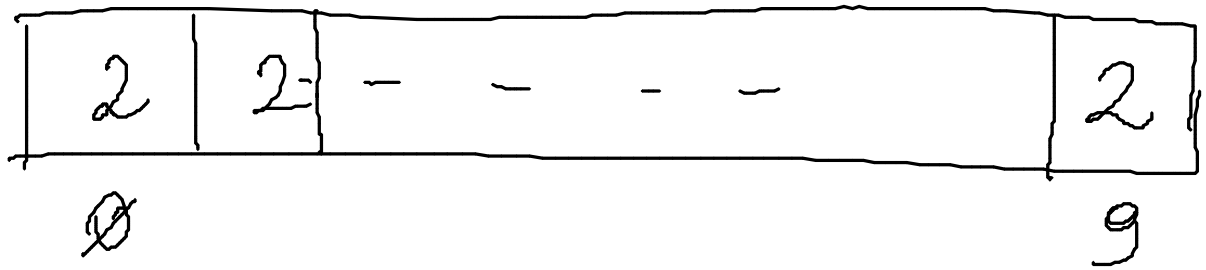
```
{ int a[10]; int x = 20;
  .... leggiamo i valori dell' array ....
  { int i = 0;
    while (i < 10 && a[i] != x)
      i = i + 1;
```

$A \wedge B$

$\equiv$

$B \wedge A$

~~while (a[i] != x && i < 10)~~
~~i = i + 1;~~

a 

i	x = 20
0	
1	
2	
⋮	
9	
10	

```
while ((i < 10) && a[i] != x)
    i = i+1;
```

Il C valuta A1 && A2 && ... && Ak
da sinistra a destra e non appena trova
che uno dei membri della congiunzione è FALSE
non procede e restituisce FALSE

$(\underline{A1}\ \&\&\ \underline{A2}\ \&\&\ \ldots\ \underline{\&\&}\ AK)\ ||\ B$

B1 || B2 valuta B1 : se B1 è TRUE restituisce TRUE senza valutare B2 altrimenti valuta B2

```
while (a[i] != x && i < 10)
    i = i+1;
```

i
0
⋮
10

```
int i = 0; int trovato = 0;
while (i < 10 && !trovato)
    if (a[i] == x) trovato = 1;
    else i = i+1;
```

Tutte le volte che scriviamo del codice fatto in questo modo

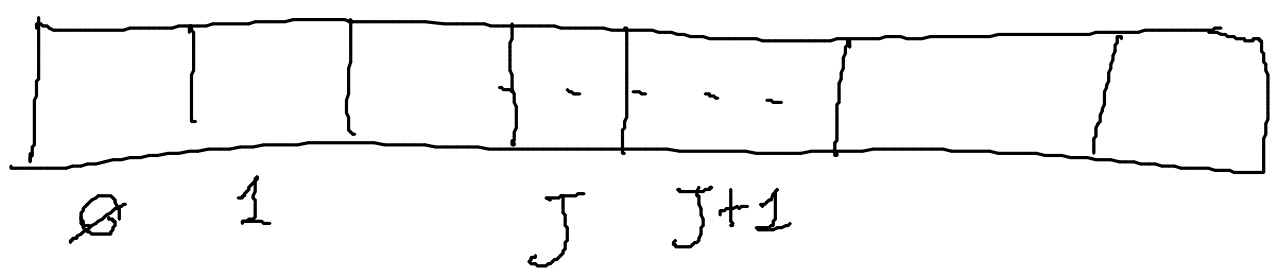
```
while ( guardia )
{ corpo ... a[i]
.....
a[exp]
. . . .
}
```

dobbiamo garantire che la guardia assicuri che i valori delle espressioni usate come indice dell'array siano compresi nell'intervallo $[0, M)$ dove M è la dim. dell'array

```
{ int a[10]; int i; int ordinato;
.... leggiamo i valori di a ....
  i = 0; ordinato = 1;
  while (i < 9 && ordinato)
    if (a[i] <= a[i+1]) i = i+1;
        else ordinato = 0;
```

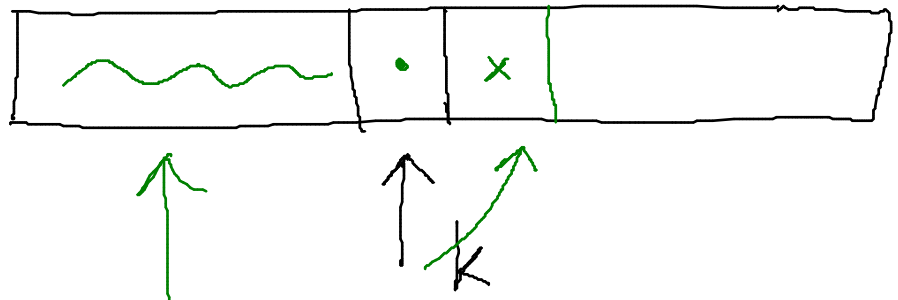
J = 9

J+1 = 10



$\forall j \in [0, 9) . \; a[j] <= a[j+1]$

• <= x



questa porzione di array
è ORDINATA

i	ordinato
∅	true
⋮	
k	true
⋮	⋮

guarda
iterazione del
ciclo, quando i = k e
ci accingiamo a valutare la
guardia

(i < 9 && ordinato)

PROPRIETÀ DEL CICLO

$$\forall_{j \in [0, i)} . \; a[j] <= a[j+1]$$

≡

ordinato

AUMENTARE di 1 tutti gli elementi dell'array di dimensione N

(map incr1 l)

```
{ int i = 0;
  while (i < N)
  { a[i] = a[i] + 1;
    i = i + 1;
  }
}
```

inizializzazione della VARIABILE DI CONTROLLO

valore massimo che può assumere la variabile di controllo

operazione da applicare a tutti gli elementi

incremento della variabile di controllo

ITERAZIONE DETERMINATA

```
for ( i = 0; i < N; i = i + 1 )
    a[i] = a[i] + 1;
```

"per tutti gli indici compresi in [0, N) aumenta di 1 il corrispondente elemento dell'array, scandendo gli indici da uno in uno"

```
for (i=0; i<N; i=i+2)
    a[i] = a[i] + 1;
```

da due in due - ...

è equivalente a :

```
i = 0;
while (i<N)
   { a[i] = a[i] + 1;
     i = i + 2;
   }
```

ITERAZIONE DETERMINATA

```
for ( c ; e ; c' )
      c''
```

è equivalente a:

```
c ;
while (e)
   { c'' ;
     c' ;
   }
```

CALCOLARE LA SOMMA DEI VALORI DI UN ARRAY

```
int somma = 0; int i;
for (i = 0; i < N; i = i+1)
    somma = somma + a[i];
```

$$somma = \sum_{j=0}^{i-1} a[j]$$

PROPRIETÀ INVARIANTE

alla prima iterazione

$$somma = \sum_{j=0}^{-1} a[j] = 0$$

in fondo

$$somma = \sum_{j=0}^{N-1} a[j]$$

PROCEDURE e FUNZIONI su ARRAY

```
void incr (int a[], int dim)
{ int i;
  for (i=0; i<dim; i=i+1)
      a[i] = a[i] + 1;
}
```

```
{  int  v[10];
   int  b[45];
    ⋮
   incr(v, 10);
    ⋮
```

```
incr(b, 45);  ⟶  incr(b, 20);
⋮                 ↑
⋮                 ↑
```

FUNZIONE CHE CALCOLA IL MASSIMO DI UN ARRAY

```
int massimo (int a[], int dim)
{ int i; int m = a[0];
  for (i=1; i<dim; i=i+1)
    if (a[i] > m) m = a[i];
  return m;
}
```

```
int member (int a[], int from, int to, int x)
  /* calcola TRUE se
        ∃ j ∈ [from, to) . a[j] = x
     calcola FALSE altrimenti */
{ int trovato = 0; int i = from;
  while (i < to && ! trovato)
    if (a[i] == x) trovato = 1;
       else i = i+1;
  return trovato;
}
```

```
{  int  b[500];   int z;  int ee;
        ⋮
```

→ `ee = member(b, 0, 500, z)`

`ee = member(b, 0, 50, z)`

`ee = member(b, 450, 500, z)`