Dati due array, a e b, di dimensioni dima e dimb, rispettivamente.
Tutti gli elementi di a compaiono anche in b.

$$(\forall i \in [0, dima) . (\exists j \in [0, dimb) . a[i] = b[j]))$$

Quantificazioni con DOMINIO
(le variabile quantificate appartiene a un insieme)

$\forall i \in [0, dima) .$

$\exists j \in [0, dimb) .$

Quantificazione universale con dominio

$$(\forall i \in I . P) \equiv (\forall i . i \in I \Rightarrow P)$$

quantificazione senza dominio

perché è nella forma

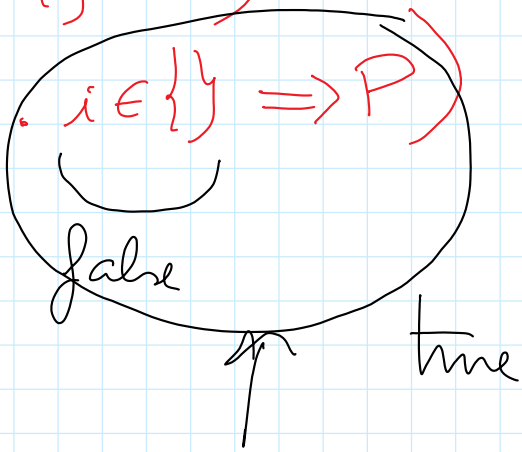
perché è nella forma

$$(\forall i.\ A \Rightarrow B)$$

Quantificazione universale con dominio vuoto quanto vale? true!!

$$(\forall i \in \{\}.\ P) \equiv$$

$$(\forall i.\ i \in \{\} \Rightarrow P)$$



$(\exists i \in I \,.\, P) \equiv$

quantificazione esistenziale con dominio

$(\exists i.\ i \in I \wedge P)$

Quanto vale la quantificazione esistenziale con dominio vuoto? false

$(\exists i \in \{\} \,.\, P) \equiv$

$(\exists i.\ i \in \{\} \wedge P)$

false

esiste un elemento ....

inizialmente, quando dobbiamo ancora controllare gli elementi dell'array, è come se avessimo un dominio vuoto

trovato = ∅ ;

tutti gli elementi ...

//

ok = true ;

$$\left(\forall i \in [0, dima) \cdot \left(\exists j \in [0, dimb) \cdot a[i] = b[j]\right)\right)$$

Member

```
int member (int el, int a[], int dim)
{ int i = 0;
  int trovato = 0;
  while (i < dim && !trovato)
     if (el == a[i]) trovato = 1;
     else i++;
  return trovato;
}
```

```
int tuttiinb (int a[], int dima,
              int b[], int dimb)
{ int i = 0;
  int ok = 1;
  while (i < dima && ok)
    if (member (a[i], b, dimb)) i++;
    else ok = 0;
  return ok;
}
```

```
if (! member (a[i], b, dimb)) ok = 0;
else i++;
```

Cardinalità di un insieme I

$\#I$ è il numero degli elementi di I se I è un insieme finito.

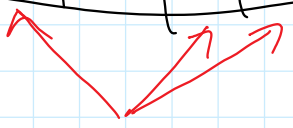
$$\#\{0,1,2\} = 3$$

$$\#\{a,b\} = 2$$

Vogliamo controllare che, nell'array a, esiste un elemento che compare esattamente 3 volte

es:

a | 2 | 1 | 5 | 1 | 1 | 5 | → true → 1



a | 2 | 1 | 5 | 1 | 1 | 1 | → false → 0

$(\exists i \in [0, dim) .$

$\#\{ j \mid j \in [0, dim) \wedge a[j] = a[i] \} = 3)$

insieme degli indici degli elementi di a uguali all'elemento a[i].

```
int conta (int el, int a[], int dim)
{ int i;
  int c = 0;
  for (i = 0; i < dim; i++)
      if (a[i] == el) c++;
  return c;
}
```

Conta quante volte el compare in a

```
int esattamente3(int a[], int dim)
{ int i=0;
  int trovato=0;
  while(i<dim && !trovato)
    if(conta(a[i], a, dim)==3) trovato=1;
     else i++;
  return trovato;
}
```

Tutti gli elementi di a compaiono esattamente 3 volte.

$$(\forall i \in [0, dim).\ \#\{j \mid j \in [0, dim) \wedge a[j] = a[i]\} = 3)$$

```
int esattamente3 (int a[], int dim)
{ int i = 0;
  int ok = 1;
  while (i < dim && ok)
    if (conta (a[i], a, dim) != 3) ok = 0
    else i++;
  return ok;
}
```

```
if (conta (a[i], a, dim) == 3) i++;
else ok = 0;
```

Trovare il valore massimo di un array.

es:

a | -2 | 3 | 7 | 5 | 12 | -3 | ⟶ 12

a | -2 | 12 | 7 | 5 | 12 | -3 | ⟶ 12

m = ~~-2~~ 12

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

Vogliamo due risultati:
- il valore massimo
- il suo indice

a

7	-2	10	-3	-4	2
0	1	2	3	4	5

max → 10
indice → 2

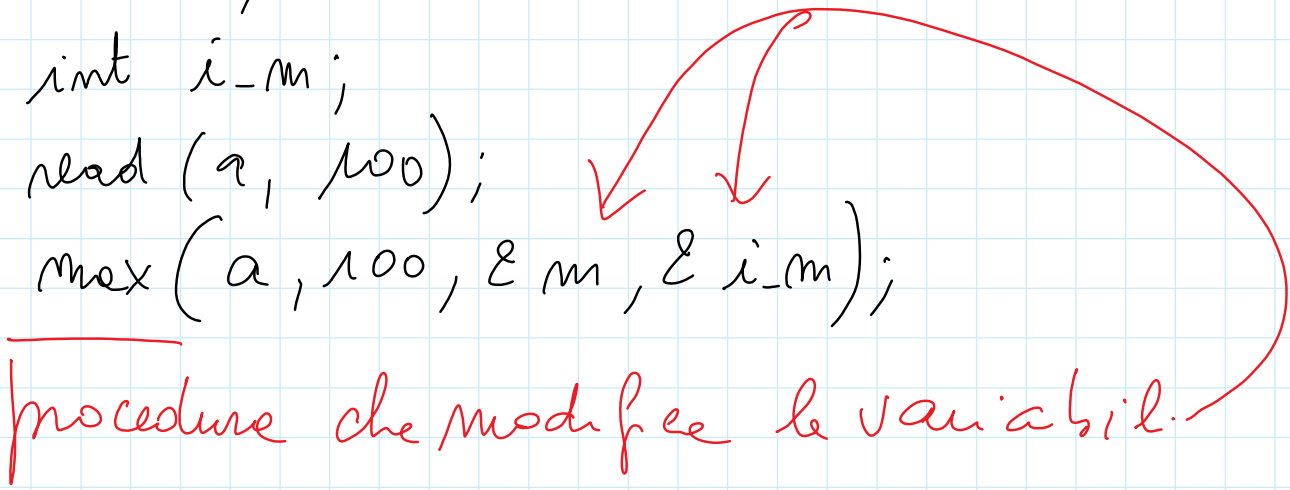
```
int max (int a[], int dim, int *ind_max)
{
   ...
}

main()
{ int a[100];
  int m;
  int i_m;
  read(a, 100);
  m = max(a, 100, &i_m);
```

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

Nel caso in cui ci siano più occorrenze del valore massimo si deve restituire quale indice? del primo!

>= dell'ultimo!

```
main()
{ int a[100];
  int m;
  int i_m;
  read(a, 100);
  max(a, 100, &m, &i_m);
```

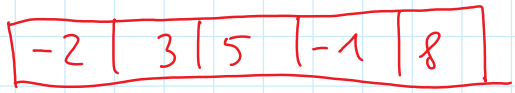


procedure che modifica le variabili

```
void max(int a[], int dim, int *m, int *i_m)
{ int i;
  *m = a[0];
  *i_m = 0;
  for(i=1; i<dim; i++)
    if(a[i] > *m)
      { *m = a[i];
        *i_m = i;
      }
}
```

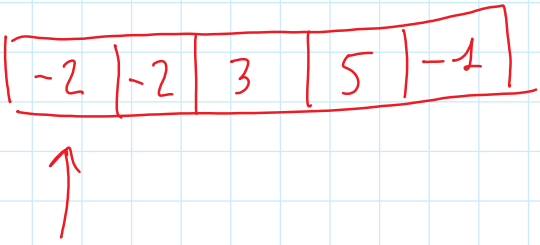
SHIFT di un array

a

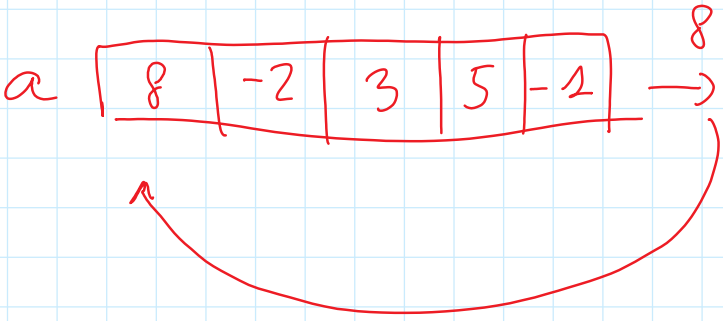


shift verso destra

a



a



shift circolare verso destra

shift circolare verso destra.

mercoledì 2 ottobre 2019 12:43

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

NO!

5 -1 -1 -1

temp = 5

a

-1	2	3	5
0	1	2	3

5 -1 2 3

temp = 5

a

-1	2	3	5

```
void shift (int a[], int dim)
{ int temp = a[dim-1];
  int i;
  for( i=dim-1 ; i>0 ; i--)
       a[i] = a[i-1];

  a[0] = temp;
}
```

i = i-1;

mercoledì 2 ottobre 2019 12:51

Shift circolare verso destra di n posizioni

↳ 3 posizioni

a | 1 | 2 | 3 | 4 | ⟶ | 2 | 3 | 4 | 1 |

↳ 7 posizioni

```
void shiftn (int n, int a[], int dim)
{ int i;
  for(i=0; i<n; i++)
      shift(a, dim);
}
```

Cercare l'elemento massimo e l'elemento minimo di un array.

a | 3 | 3 | 3 |

```
Void maxmin (int a[], int dim,
             int* max, int* min)
{ int i;
  *max = a[0];
  *min = a[0];
  for (i=1; i<dim; i++)
      if (a[i] > *max) *max = a[i];
      else if (a[i] < *min) *min = a[i];
}
main()
{ int a[100];
  int max, min;
  read (a, 100);
  maxmin (a, 100, &max, &min);
```