



Array VETTORE

Marco Alberti

998 []



Dipartimento
di Matematica
e Informatica



Università
degli Studi
di Ferrara

Programmazione e Laboratorio, A.A. 2020-2021

Ultima modifica: 7 novembre 2020

Attenzione! Questo materiale didattico è per uso personale dello studente ed è coperto da copyright.
Ne sono vietati la riproduzione e il riutilizzo anche parziale, ai sensi e per gli effetti della legge sul diritto d'autore.

Sommario

1 Array

FILTER
MAP

2 Liste, sequenze e pattern con array

SEQUENZIALI

3 Ordine e ordinamento

Sommario

1 Array

2 Liste, sequenze e pattern con array

3 Ordine e ordinamento

Esercizio

Scrivere un programma che richieda all'utente cinque numeri interi e li stampi in ordine inverso. Ad esempio, se l'utente digita i numeri 5, 3, 7, 8 e 2, il programma deve stampare

- a 2
- d 8
- c 7
- b 3
- a 5

Quante variabili servono? 5

int i; int n;
for(i=0, i<=5, i++)
 scanf ("%d", &n);

for (i = 0, i <= 5; i++)
 printf ("%d\n", n);

n 54782

Una soluzione

090_array/cinque.c

```
1 #include <stdio.h>
2
3 main() {
4     int a, b, c, d, e;
5     scanf("%d", &a);
6     scanf("%d", &b);
7     scanf("%d", &c);
8     scanf("%d", &d);
9     scanf("%d", &e);
10
11    printf("%d\n", e);
12    printf("%d\n", d);
13    printf("%d\n", c);
14    printf("%d\n", b);
15    printf("%d\n", a);
16 }
```

Funziona, ma

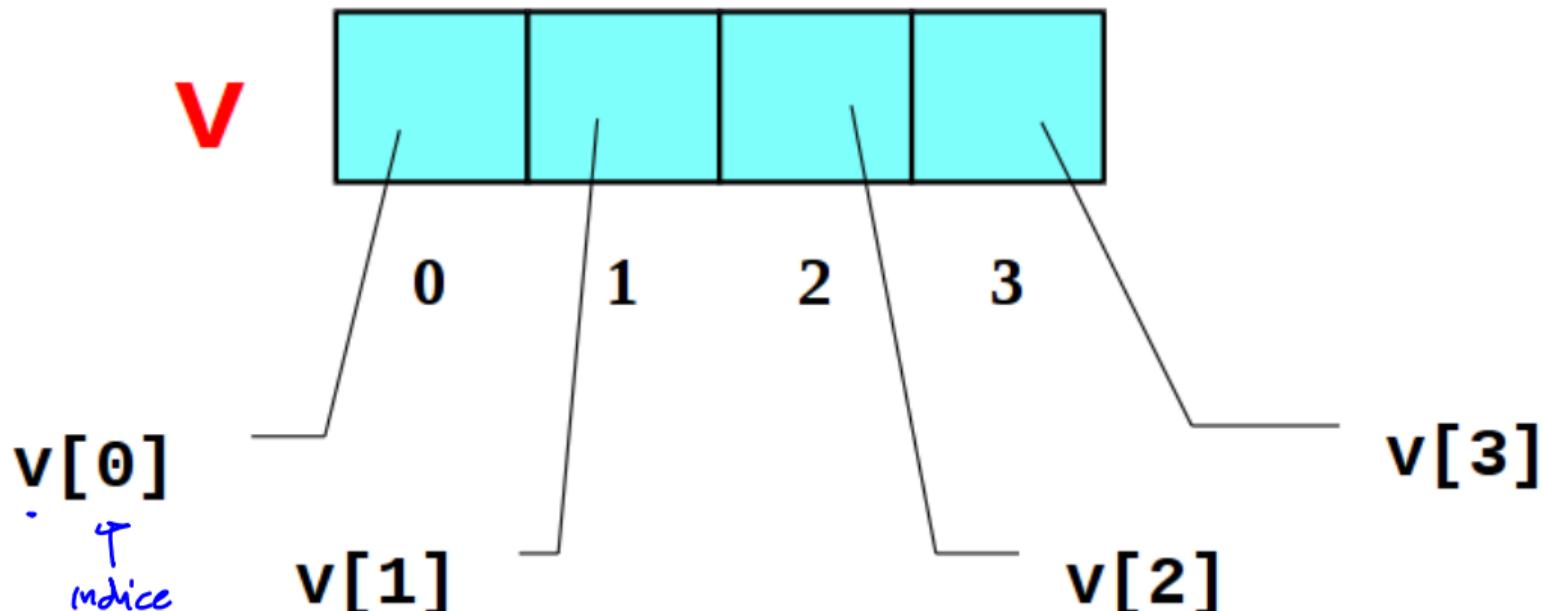
- ho dovuto scrivere codice ripetitivo
- e se i numeri fossero 100? Dovrei
 - Inventare 100 nomi di variabili (poco male: potrei usare **a1**, **a2**, ..., **a100**)
 - Definirle tutte (e questo è laborioso e soggetto a errori)
 - Scrivere 100 chiamate a **scanf** e 100 chiamate a **printf**: laborioso e soggetto a errori!

NON SCALA

Array

ELEMENTI

Un **array** (o **vettore**) di N elementi è una collezione finita di N variabili dello stesso tipo, ognuna identificata da un indice compreso fra 0 e $N - 1$.



Definizione di array

int

v

4

.

$\langle \text{definizioneArray} \rangle ::= \langle \text{tipo} \rangle \langle \text{identificatore} \rangle [\langle \text{dimensione} \rangle] ;$

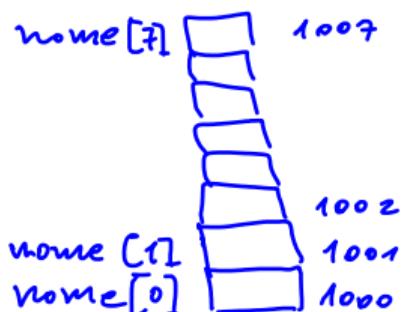
La definizione crea in memoria $\langle \text{dimensione} \rangle$ celle consecutive di tipo $\langle \text{tipo} \rangle$.



Esempio

int v[4]; crea n array di 4 variabili di tipo intero.

- *char nome[2*4];* crea un array di 8 variabili consecutive di tipo *char*.



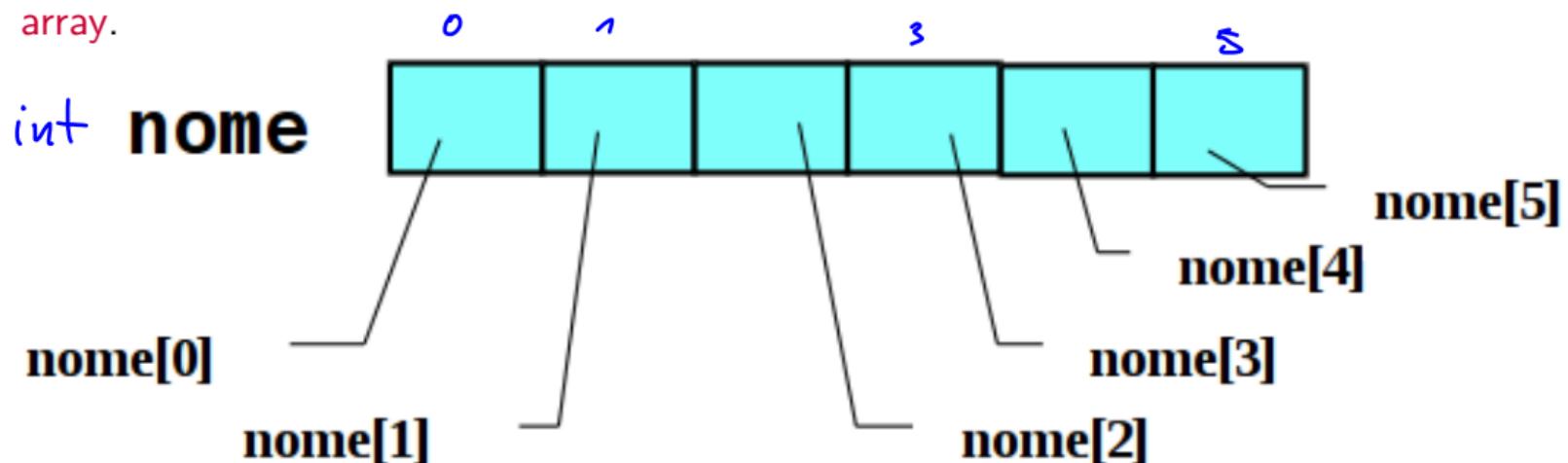
Identificazione di elementi di array

6

Ogni elemento di un array di dimensione N è identificato da un indice compreso fra 0 e $N - 1$.

nome *s*
 $\langle \text{elemento} \rangle ::= \langle \text{identificatore} \rangle [\langle \text{espressioneIntera} \rangle]$

Le espressioni che identificano un elemento sono anche dette espressioni di **selezione da array**.



Come si definisce l'array in figura?

int *nome*[6];

Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```



numeri[4]	4294952620 1448436443
numeri[3]	4294952616 -14484
numeri[2]	4294952612 -14492
numeri[1]	4294952608 1
numeri[0]	4294952604 5

main (6)

Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]); 7
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

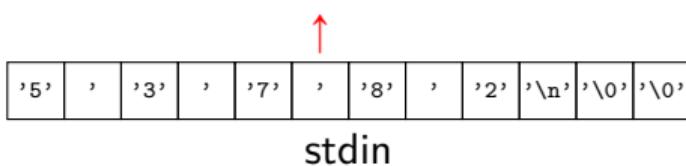


numeri[4]	4294952620 1448436443
numeri[3]	4294952616 -14484
numeri[2]	4294952612 -14492
numeri[1]	4294952608 3
numeri[0]	4294952604 5

main (7)

Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

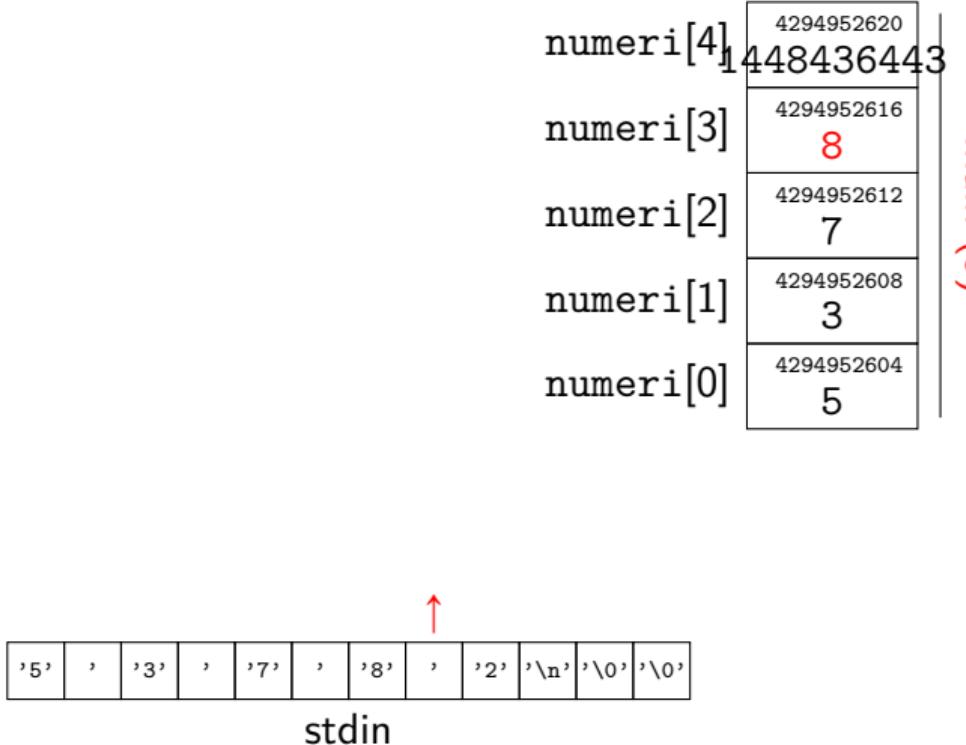


numeri[4]	4294952620 1448436443
numeri[3]	4294952616 -14484
numeri[2]	4294952612 7
numeri[1]	4294952608 3
numeri[0]	4294952604 5

main (8)

Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```



Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

numeri[4]	4294952620
	2
numeri[3]	4294952616
	8
numeri[2]	4294952612
	7
numeri[1]	4294952608
	3
numeri[0]	4294952604
	5

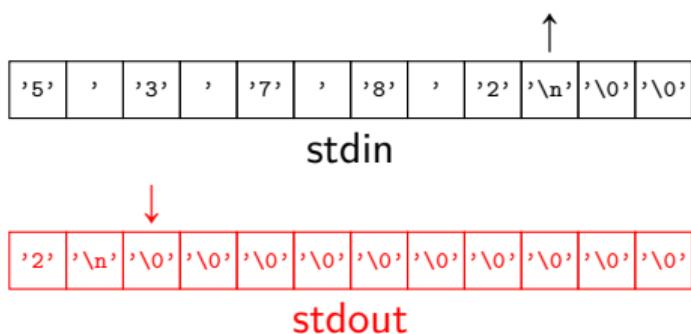


Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

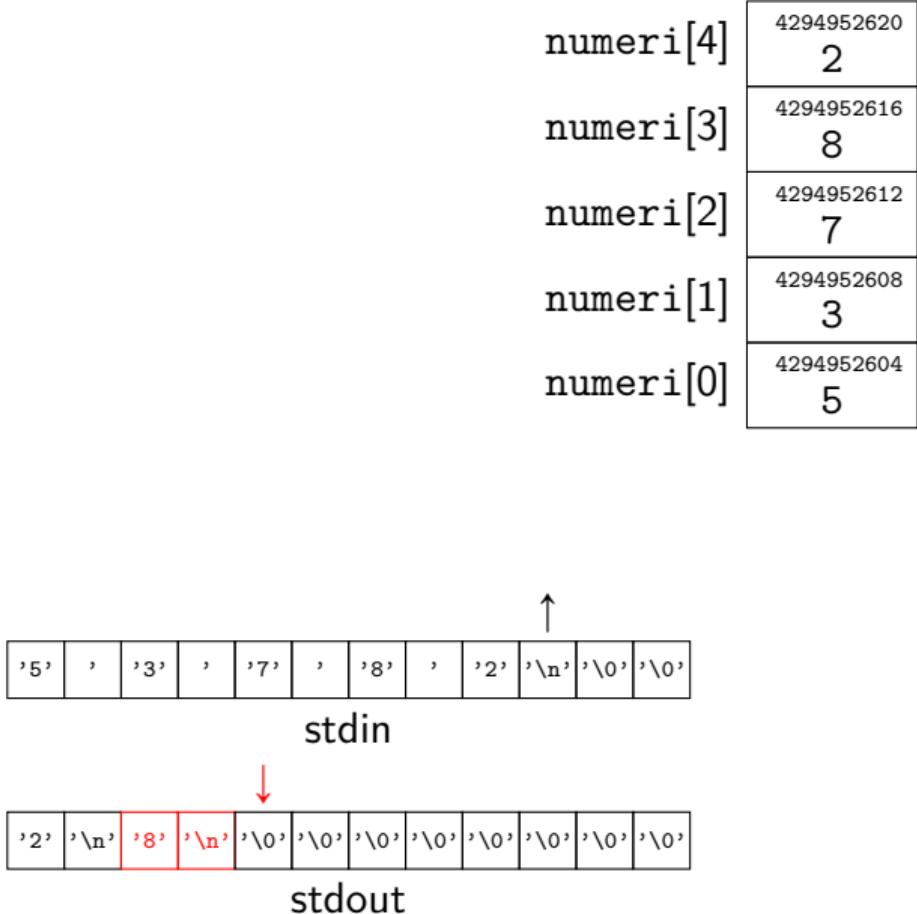
numeri[4]	4294952620
numeri[3]	2
numeri[2]	4294952616
numeri[1]	8
numeri[0]	4294952612
	7
	4294952608
	3
	4294952604
	5

main (12)



Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

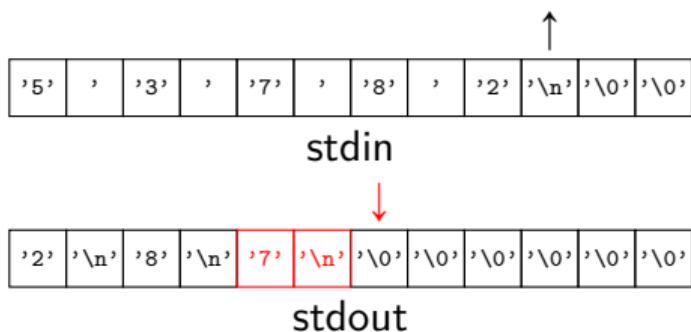


Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

numeri[4]	4294952620
numeri[3]	2
numeri[2]	4294952616
numeri[1]	8
numeri[0]	4294952612
	7
	4294952608
	3
	4294952604
	5

main (14)

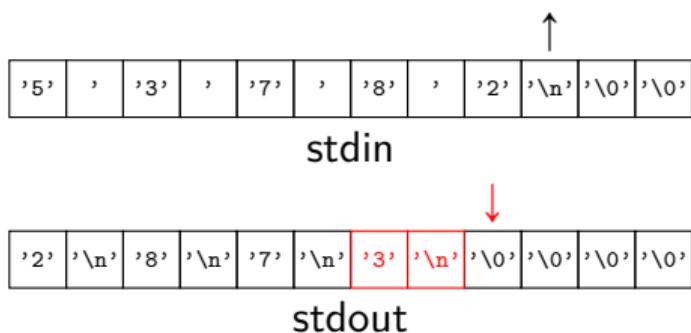


Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

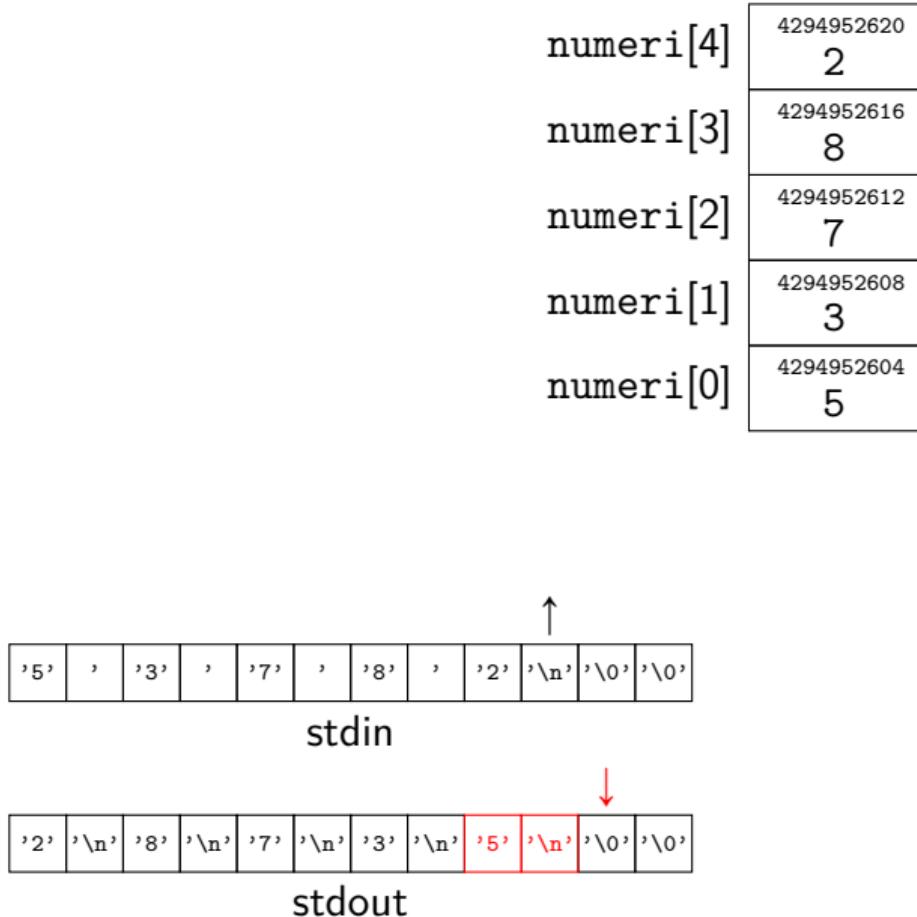
numeri[4]	4294952620
numeri[3]	2
numeri[2]	4294952616
numeri[1]	8
numeri[0]	4294952612
	7
	4294952608
	3
	4294952604
	5

main (15)



Cinque con array, vers. 1

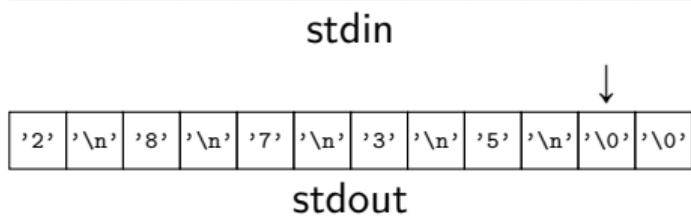
```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```



Cinque con array, vers. 1

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

4294952620	numeri[4]
2	numeri[3]
4294952616	numeri[2]
8	numeri[1]
4294952612	numeri[0]
7	
4294952608	
3	
4294952604	
5	



main (17)

Soluzione con array

090_array/cinque-a1.c

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     scanf("%d", &numeri[0]);
6     scanf("%d", &numeri[1]);
7     scanf("%d", &numeri[2]);
8     scanf("%d", &numeri[3]);
9     scanf("%d", &numeri[4]);
10
11    printf("%d\n", numeri[4]);
12    printf("%d\n", numeri[3]);
13    printf("%d\n", numeri[2]);
14    printf("%d\n", numeri[1]);
15    printf("%d\n", numeri[0]);
16
17    return 0;
18 }
```

Ho risolto

- il non-problema della scelta nomi delle variabili
- il problema della potenziale definizione di molte variabili

Ma mi rimane quello più grave: la ripetizione del codice!

Che cosa cambia fra una chiamata a **scanf** e le altre, e fra una **printf** e le altre? Solo l'indice dell'elemento dell'array.

Espressioni per indice

numeri[3]

```
int i;
for (i = 0, i < 5, i++)
printf numeri[i],
```

- Ciò che rende gli array utili ad evitare ripetizioni è che l'indice non deve essere una costante intera, ma una qualunque espressione di valore intero: variabile, operazione intera, chiamata di funzione a valore intero...
- Nell'esempio alla slide 6, gli elementi selezionati si possono ottenere usando un'espressione variabile come indice, e facendo variare opportunamente la variabile dell'espressione.

numeri[i];

- La sequenza dei valori dell'indice determina la sequenza degli elementi dell'array utilizzati nelle espressioni.

int a[10];

Ad esempio: se, nell'espressione di output `printf("%d", a[i])`, `i` assume la sequenza di valori 1, 2, 5, verranno stampati gli elementi `a[1]`, `a[2]` e `a[5]`.

- Conosciamo qualche tecnica per far assumere a una variabile come un indice una sequenza di valori che ci fa comodo?

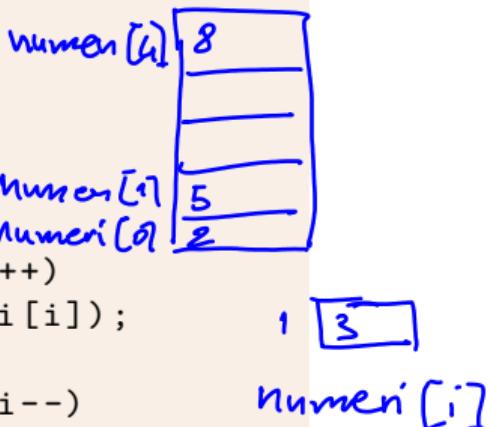
Soluzione migliore

Nel programma alla slide 6 devono essere selezionati gli elementi dell'array con indici che vanno da 0 a 4, e poi da 4 a 0; si possono selezionare gli elementi dell'array usando come indice la variabile di controllo di due cicli **for**.

090_array/cinque-a2.c

```

1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```



- Una sola chiamata a **scanf** e una sola chiamata a **printf**
- Se cambia il numero degli interi basta modificare la dimensione dell'array e le condizioni di permanenza dei cicli **for**.

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

Diagram showing the state of the array numeri during execution:

- Index 0: 0
- Index 1: 1
- Index 2: 2
- Index 3: 3
- Index 4: 4

numeri[4]	4294952568 -14532
numeri[3]	4294952564 -14540
numeri[2]	4294952560 1
numeri[1]	4294952556 1448436355
numeri[0]	4294952552 -14532
i	4294952572 1448436315

main(6)

Cinque con array, vers. 2

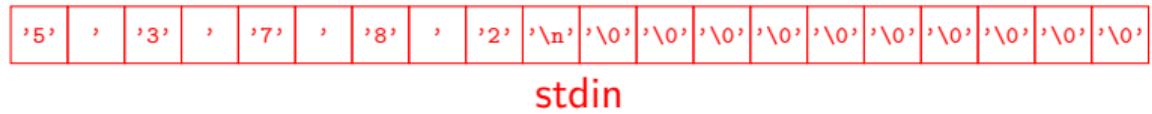
```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8         0
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

numeri[4]	4294952568 -14532
numeri[3]	4294952564 -14540
numeri[2]	4294952560 1
numeri[1]	4294952556 1448436355
numeri[0]	4294952552 -14532
i	4294952572 0

main (7)

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```



numeri[4]	4294952568
numeri[3]	-14532
numeri[2]	4294952564
numeri[1]	-14540
numeri[0]	1
i	4294952556
numeri[1]	1448436355
numeri[0]	4294952552
i	5
numeri[1]	4294952572
i	1

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

stdin

'5'	,	'3'	,	'7'	,	'8'	,	'2'	'\n'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
-----	---	-----	---	-----	---	-----	---	-----	------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

numeri[4]	4294952568
numeri[3]	-14532
numeri[2]	4294952564
numeri[1]	-14540
numeri[0]	4294952560
i	1
	4294952556
	3
	4294952552
	5
	4294952572
	2

main (7)

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

numeri[4]	4294952568 -14532
numeri[3]	4294952564 -14540
numeri[2]	4294952560 7
numeri[1]	4294952556 3
numeri[0]	4294952552 5
i	4294952572 3

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

stdin

4294952568
-14532
4294952564
8
4294952560
7
4294952556
3
4294952552
5
4294952572
4

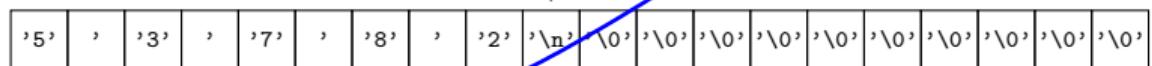
Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

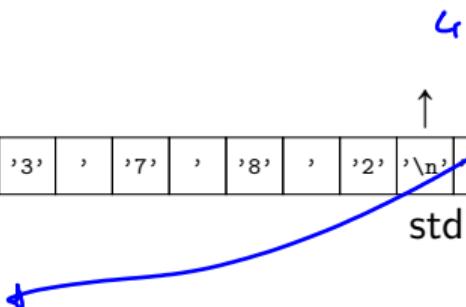
numeri[4]	4294952568 2
numeri[3]	4294952564 8
numeri[2]	4294952560 7
numeri[1]	4294952556 3
numeri[0]	4294952552 5
i	4294952572 5

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```



stdin

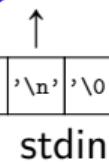


numeri[4]	4294952568 2
numeri[3]	4294952564 8
numeri[2]	4294952560 7
numeri[1]	4294952556 3
numeri[0]	4294952552 5
i	4294952572 4

main (10)

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

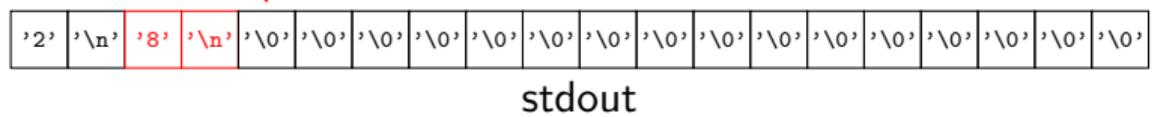
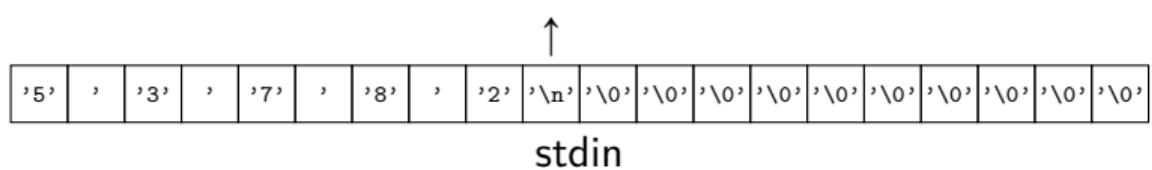


numeri[4]	4294952568 2
numeri[3]	4294952564 8
numeri[2]	4294952560 7
numeri[1]	4294952556 3
numeri[0]	4294952552 5
i	4294952572 3

main (10)

Cinque con array, vers. 2

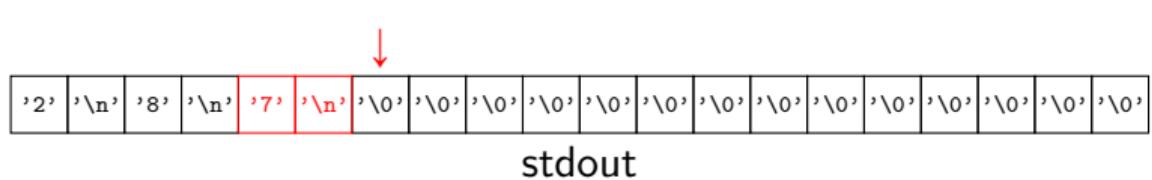
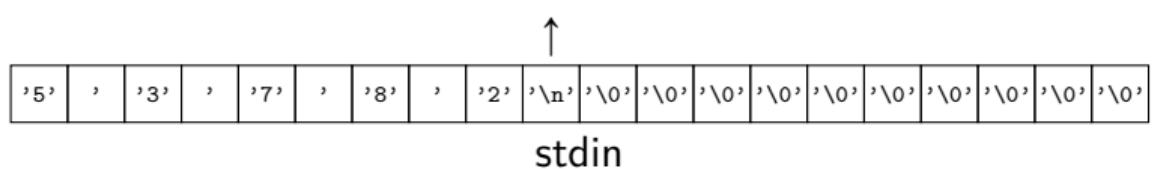
```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```



numeri[4]	4294952568
numeri[3]	2
numeri[2]	4294952564
numeri[1]	8
numeri[0]	4294952560
i	7
main (10)	4294952556
	3
	4294952552
	5
	4294952572
	2

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

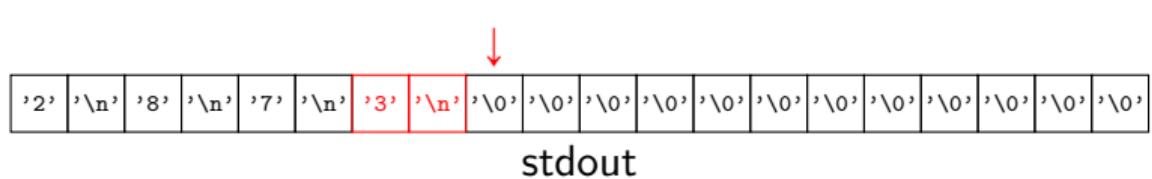
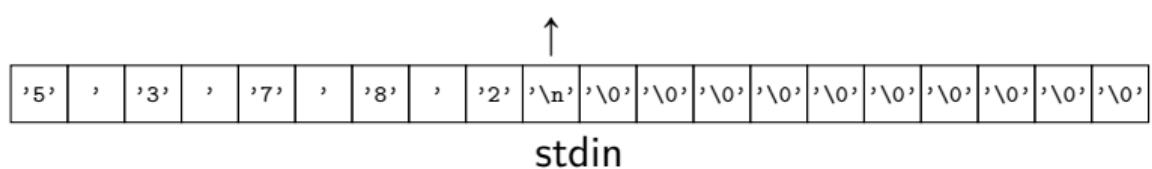


numeri[4]	4294952568 2
numeri[3]	4294952564 8
numeri[2]	4294952560 7
numeri[1]	4294952556 3
numeri[0]	4294952552 5
i	4294952572 1

main (10)

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```

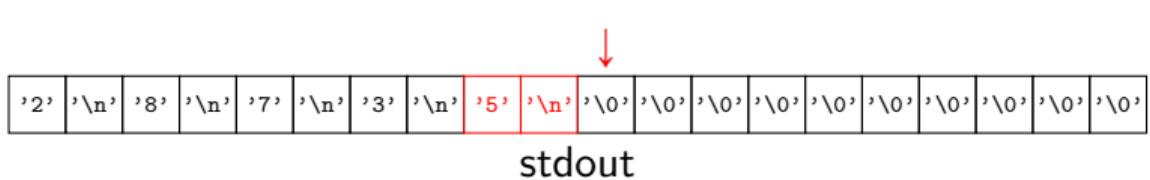
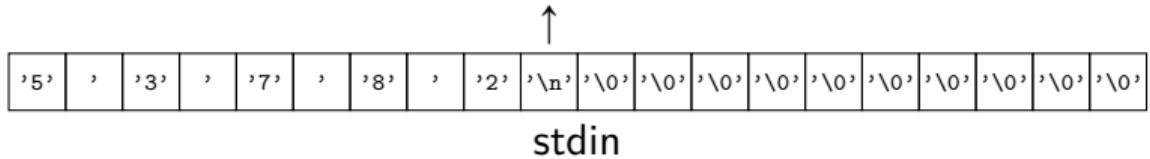


numeri[4]	4294952568 2
numeri[3]	4294952564 8
numeri[2]	4294952560 7
numeri[1]	4294952556 3
numeri[0]	4294952552 5
i	4294952572 0

main (10)

Cinque con array, vers. 2

```
1 #include <stdio.h>
2
3 int main(void) {
4     int numeri[5];
5     int i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", &numeri[i]);
8     5-1
9     for (i = 4; i >= 0; i--)
10        printf("%d\n", numeri[i]);
11 }
```



numeri[4]	4294952568
numeri[3]	2
numeri[2]	4294952564
numeri[1]	8
numeri[0]	4294952560
i	7
	4294952556
	3
	4294952552
	5
	4294952572
	-1

main (11)

Eliminare ripetizione dimensione

Nel programma alla slide 8 la dimensione dell'array compare tre volte (dove?). Se cambiasse, bisognerebbe modificare il programma in tre punti. Perché non mettere la dimensione in una variabile?

090_array/cinque-var.c

```
1 #include <stdio.h>      DIM=10;    DIM 5
2
3 int main(void) {
4     int DIM;
5     scanf("%d", &DIM); → SBAGLIATO SECONDO STANDARD C89
6     int numeri[DIM];
7     int i;
8     for (i = 0; i < DIM; i++)
9         scanf("%d", &numeri[i]);
10
11    for (i = DIM - 1; i >= 0; i--)
12        printf("%d\n", numeri[i]);
13    return 0;
14 }
```

Macro

GCC compila, ma

- secondo lo standard C89, la dimensione di un array deve essere costante¹;
- anche negli standard successivi (da C99 in poi) i compilatori C non sono tenuti a implementare gli array di dimensione variabile.

Il problema si può aggirare con l'uso di una **macro**:

$\langle\text{defineMacro}\rangle ::= \#define \langle\text{stringa1}\rangle \langle\text{stringa2}\rangle$

DIRETTIVA

Ogni occorrenza di $\langle\text{stringa1}\rangle$ nel programma viene sostituita con $\langle\text{stringa2}\rangle$; quindi $\langle\text{stringa1}\rangle$ funziona come una sorta di variabile testuale.

Questa operazione viene svolta dal cosiddetto **preprocessore C** (comando **cpp**; viene comunque chiamato da **gcc**) prima della compilazione.

```
#include <stdio.h>
int numeri [DIM],
```

¹Per far mostrare a **gcc** la discrepanza rispetto allo standard, si può compilare il file con le opzioni **-std=c89** e **-pedantic**.

Eliminazione della ripetizione con macro

090_array/cinque-macro.c

```
1 #include <stdio.h>
2 #define DIM 5 ← DIRETTIVA
3
4 int main(void){1a
5     int numeri[DIM];
6     int i;
7     for (i = 0; i < DIM; i++)1a
8         scanf("%d", &numeri[i]);
9
10    for (i = DIM - 1; i >= 0; i--)1a
11        printf("%d\n", numeri[i]);
12
13    return 0;
14 }
```

Inizializzazione di array

`int a[5], a[0]=2; a[1]=4; ...`

Un array può essere inizializzato nella sua definizione, mettendo gli elementi fra parentesi graffe:

~~`int a[5]; a = {2, 4, 6, 1, 2};`~~

Esempio

`int a[5] = {2, 4, 6, 1, 2};`

Nella definizione di array inizializzati si può omettere la dimensione (perché il compilatore ha le informazioni per capirla da solo):

<code>a[4]</code>	2
<code>a[3]</code>	1
<code>a[2]</code>	6
<code>a[1]</code>	4
<code>a[0]</code>	2

Esempio

`int a[] = {2, 4, 6, 1, 2};`

Stampa array inizializzato

Scrivere un programma che definisca un array inizializzato come negli esempi in questa diapositiva, e poi ne stampi gli elementi, uno su ogni riga.

Accesso a elementi estranei all'array

OUTPUT

2 5 3 4 1 0 100

Che cosa fa questo programma?

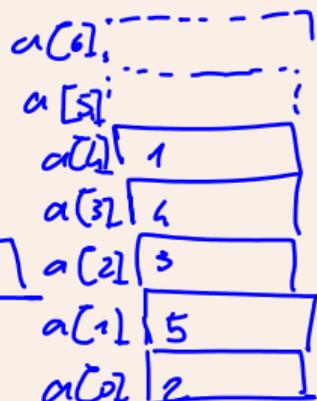
090_array/bounds.c

```

1 #include <stdio.h>
2
3 int main() {
4     int a[] = {2, 5, 3, 4, 1}, i = 0;
5     while (1) {
6         printf("%d\n", a[i]);
7         i++;
8     }
9 }
```

indice

i 0 1 2 3 4 5 6



E' responsabilità del programmatore assicurarsi che l'indice di una selezione sia fra quelli ammissibili (cioè fra 0 e la dimensione dell'array). La macchina astratta tenta sempre di accedere all'elemento selezionato; se questo si trova fuori dallo spazio di indirizzamento del programma, il sistema operativo termina il programma con un **errore di segmentazione**, o **segmentation fault**.

Esercizio (da prova teorica del 21/1/2019)

Detta m la rappresentazione come intero decimale del proprio numero di matricola, la funzione di prototipo `int f(int d)` restituisce

- la d -esima cifra di m a partire da sinistra se d è compreso fra 1 e il numero di cifre di m ; $f(-1) \rightarrow 0$
 $f(1) \rightarrow 0$
- 0 altrimenti. $f(4) \rightarrow 3$ $f(0) \rightarrow 0$

Per esempio, se il numero di matricola è **654321**, la chiamata `f(2)` restituisce **5**, mentre le chiamate `f(0)` e `f(7)` restituiscono **0**.

Quali caratteri stampa il programma costituito dal codice a fianco e dalla definizione della funzione `f`? Motivare la risposta.

23273 $f(5) \rightarrow 3$

090_array/20190121-t1.c

```

1 #include <stdio.h>
2
3 int f(int d); DICHIARAZIONE
4
5 int main() {
6     int a = 0, i[] = {1, 1, 1};
7     while (a < 2)
8         i[a] = f(++a);
9
10    for (i[0] = 2; i[0] >= 0; i
11        [0]--)
12        printf("%d ", i[i[0]]));
13    printf("\n");
14    return 0;
}

```

Codice per l'esercizio alla slide 14

23243

090_array/20190121-t1.c

```

1 #include <stdio.h>
2
3 int f(int d);
4
5 int main() {
6     int a = 0, i[] = {1, 1, 1};
7     while (a < 2)
8         i[a] = f(++a); f(1) → 2
9     int curr
10    for (i[0] = 2; i[0] >= 0; i
11        [0]--) update
12        printf("%d ", i[i[0]]); i[2] → 3
13    printf("\n");
14    return 0;
}

```

OUTPUT

3 2 0 ↶

$f(0) \rightarrow 0$
 $f(1) \rightarrow 2$
 $f(2) \rightarrow 3$
 $f(3) \rightarrow 2$
 $f(4) \rightarrow 7$
 $f(5) \rightarrow 3$
 $f(6..) \rightarrow 0$

a	0x2
i[2]	13
i[1]	12
i[0]	12 4 0 -1

Sommario

1 Array

LISTE SEQUENZIALI

2 Liste, sequenze e pattern con array

[2, 4, 8]	a[2]	8
	a[1]	4
	a[0]	2

MAP, FILTER, REDUCE . .

3 Ordine e ordinamento

Lista

ELENCO

Rappresenta una ~~sequenza~~ di entità dello stesso tipo (ad esempio interi, caratteri, strutture, altre liste...)

Notazione *astratta* (non sintassi C)

L = [el1, el2, ..., elN]

Esempio

- ['a', 'b', 'c'] è una lista di caratteri
- [2, 5, 3] è una lista di numeri interi
- [[2,4], [1], [], [2,5,4]] è una lista di liste di interi

Liste e array

LISTA

$$\{2, 5, 3, 4\} = \{2, 3, 4\}$$

$$[2, 5, 4, 3] \neq [2, 3, 4]$$

Per sequenza si intende un **multi-insieme finito e ordinato**. **DI ELEMENTI DI TIPO FISSATO**

- **multi-insieme**: è possibile che lo stesso elemento compaia più volte ($[1, 2]$ e $[2, 1]$, $[2]$ sono entrambe liste valide (e diverse)).
- **finito**: ad ogni momento, contiene un numero finito di elementi (La successione $a_n = n : n \in \mathbb{N}$ non è una lista).
- **ordinato**: due liste con gli stessi elementi che compaiono in ordine diverso sono diverse (le liste $['a', 'b']$ e $['b', 'a']$ sono diverse). $\{2, 3\} = \{3, 2\}$ $[2, 3] \neq [3, 2]$

Gli array si prestano a rappresentare liste (ogni elemento dell'array rappresenta un elemento della lista); tale rappresentazione è detta **sequenziale**, perché gli elementi sono memorizzati in aree consecutive della memoria.

Di seguito vedremo alcuni problemi tipici che riguardano le liste e i pattern per la risoluzione di questi problemi, usando la rappresentazione sequenziale (cioè con array) delle liste.

Poichè una lista è anche una sequenza, possiamo usare i pattern che già conosciamo sulle sequenze: map, filter, reduce.

Stampa

Il seguente programma stampa tutti gli elementi di un array.

090_array/stampa.c

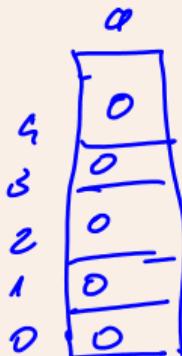
```
1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM];
6     int i;
7     for (i = 0; i < DIM; i++)
8         printf("%d\n", a[i]);
9 }
```

Inizializzazione

Il seguente programma inizializza tutti gli elementi di un array a 0.

090_array/init.c

```
1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM];
6     int i;
7     for (i = 0; i < DIM; i++)
8         a[i] = 0;
9 }
```



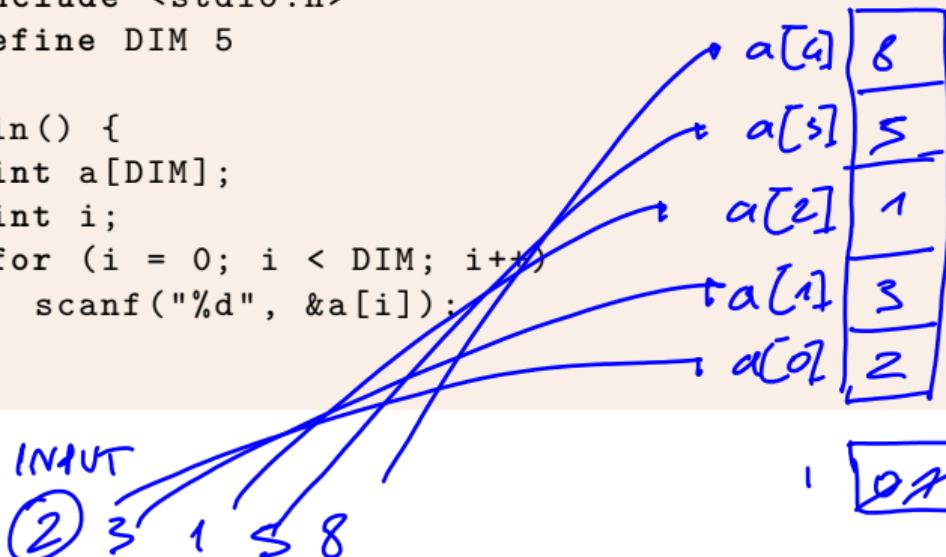
| 092865 |

Input

Che cosa fa il seguente programma?

090_array/input.c

```
1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM];
6     int i;
7     for (i = 0; i < DIM; i++)
8         scanf("%d", &a[i]);
9 }
```



1 014845

Confronto

090_array/stampa.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5   int a[DIM];
6   int i;
7   for (i = 0; i < DIM; i++)
8     printf("%d\n", a[i]);
9 }
  
```

090_array/init.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5   int a[DIM];
6   int i;
7   for (i = 0; i < DIM; i++)
8     a[i] = 0;
9 }
  
```

090_array/input.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5   int a[DIM];
6   int i;
7   for (i = 0; i < DIM; i++)
8     scanf("%d", &a[i]);
9 }
  
```

Tutti e tre i programmi eseguono un'operazione su ognuno degli elementi dell'array. L'unica cosa che cambia fra i tre programmi è l'operazione eseguita sul singolo elemento.

Pattern ForEach

Questo pattern è detto **ForEach** e i suoi elementi sono

- il vettore (o lista)
- l'**operazione** da eseguire su ciascun elemento

Si implementa con un ciclo **for**, usando una variabile per rappresentare l'indice dell'elemento dell'array nel corpo, e facendola variare su tutti i possibili indici.

090_array/foreach.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM];
6     int i;    i 0,1,2,...,DIM-1
7     for (i = 0; i < DIM; i++) {
8         // istruzione o istruzioni su a[i]
9     }
10 }
```

[] ForEach

Varianti di ForEach: direzione di scorrimento

Ovviamente nulla impedisce di scorrere gli elementi dell'array dall'ultimo al primo.

090_array/foreach-indietro.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM];
6     int i;           DIM-1, DIM-2, ..., 3, 2, 1, 0
7     for (i = DIM - 1; i >= 0; i--) {
8         // istruzione o istruzioni su a[i]   printf ("%d", a[i]);
9     }
10 }
```

In questo caso la sequenza degli elementi `a[i]` è `[a[DIM-1], a[DIM-2], ... ,a[1], a[0]]`.

Quali pattern usa il programma alla slide 9?

ForEach (input, numeri)
(inverso) ForEach (output, numeri)

Esercizio

Progressivi

Scrivere un programma che inizializzi un array di 10 elementi ai 10 numeri che seguono un numero richiesto all'utente, e poi stampi l'array.

3	15
8	14
7	13
6	12
5	11
4	10
3	9
2	8
1	7
0	6

a

n 5 !
i

$$\begin{aligned}
 & a[0] \leftarrow 6 = n+1 = n+1+i \quad \text{operazione su } a[i] \\
 & a[1] \leftarrow 7 = n+2 = n+1+i \\
 & a[2] \leftarrow 8 = n+3 = n+1+i \\
 & \quad \vdots \\
 & \quad i \\
 & a[9] \leftarrow 15 = n+10 = n+1+i
 \end{aligned}$$

for($i=0; i<10; i++$)
 $a[i] = n+1+i;$

Esercizio

Quadrati

Scrivere un programma che inizializzi un array di 10 elementi ai quadrati dei primi 10 numeri naturali, e poi stampi l'array.

$$a[9] = 10^2 = 100$$

OUTPUT

:

1 4 9 16 ... 81 100

:

$$a[8] = 8^2 = 64$$

$$a[7] = 7^2 = 49$$

$$a[6] = 6^2 = 36$$

Esercizio

Scrivere un programma che assegna ad ogni elemento di un vettore il doppio dell'elemento nella stessa posizione di un altro vettore, della stessa dimensione.

090_array/doppio.c

```

1 #include <stdio.h>
2 #define DIM 5
3 int main(void) {
4     int i, a[] = {2, 4, 1, 6, 2}, b[5];
5
6     for (i = 0; i < DIM; i++) {
7         b[i] = 2 * a[i];    ORIGINE
8     }
9     return 0;
10 }
```

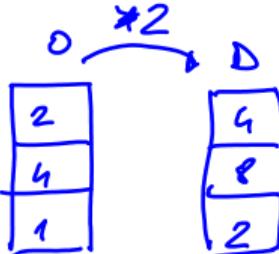
a[4]	<u>2</u>	b[4]	<u>4</u>
a[3]	<u>6</u>	b[3]	<u>12</u>
a[2]	<u>1</u>	b[2]	<u>2</u>
a[1]	<u>4</u>	b[1]	<u>8</u>
a[0]	<u>2</u>	b[0]	<u>4</u>

Pattern Map

Popola una lista di **destinazione** in modo che ogni elemento sia **funzione** dell'elemento nella stessa posizione di un'altra lista, detta di **origine**.

Gli elementi sono

- la lista di origine
- la lista di destinazione
- la funzione che determina ogni elemento di destinazione in base al corrispondente elemento di origine $f(x) = 2x$



Nella rappresentazione con array, si implementa con un ciclo **for** in cui :

- la variabile di controllo **i** assumé la sequenza di tutti gli indici validi dell'array di origine
- il corpo assegna all'elemento all'indice **i** del vettore di destinazione il risultato di un'espressione calcolata sull'elemento all'indice **i** del vettore di origine.

$$D[i] = 2 \times O[i]$$

Pattern Map

090_array/map.c

```
1 #include <stdio.h>
2 #define DIM 5
3 int main(void) {
4     int i, a[] = {2, 4, 1, 6, 2}, b[DIM];
5
6     for (i = 0; i < DIM; i++) {
7         b[i] = // espressione con a[i];
8     } ↗ DESTINAZIONE ↑ ORIGINE
9     return 0;
10 }
```

Map fra array di tipo diverso

Gli array coinvolti nel pattern map possono non essere dello stesso tipo.
Che cosa fa questo programma?

090_array/map-int-char.c

```
1 #include <stdio.h>
2 #define DIM 5
3 int main(void) {
4     int i, a[] = {2, 4, 1, 6, 2};
5     char b[5]; + ORIGINE
6     BESTINAZIONE
7     for (i = 0; i < DIM; i++) {
8         b[i] = '0' + a[i];
9     }
10    return 0;
11 }
```

]

Map

Pattern map in place

Se i dati nell'array originale non ci servono più, possiamo fare il map sullo stesso, cioè sostituire ogni elemento con una funzione di se stesso.

090_array/map-in-place.c

```

1 #include <stdio.h>
2 #define DIM 5
3 int main(void) {
4     int i, a[] = {2, 4, 1, 6, 2};
5     ORIGINE           ↗
6     for (i = 0; i < DIM; i++) {
7         a[i] = // espressione con a[i];
8     }           DESTINAZIONE ↗
9     return 0;
10 }
```

4	Z8
3	an2
2	K2
1	K8
0	Z4

d

Esercizio

1.0 2.0 4.0
+ + 9

Scrivere un programma che richieda all'utente tre coefficienti reali a , b e c e stampi le coppie ordinate $\langle x, y \rangle$, dove $y = ax^2 + bx + c$, per x che varia fra -1.0 e 1.0 a passi di 0.1 .

$$y = x^2 + 2x + 1 = (x+1)^2$$

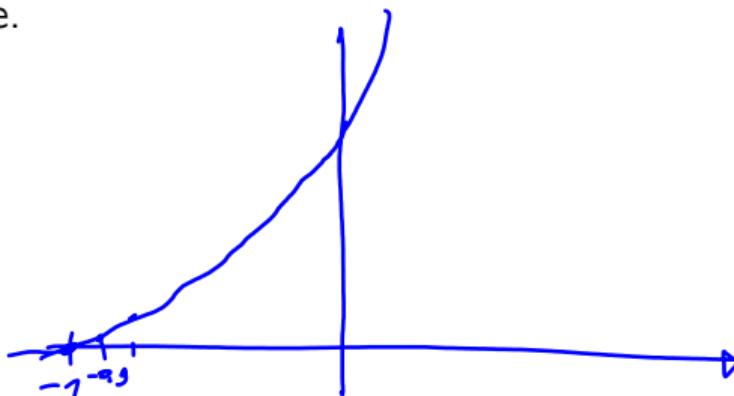
Procedere in questo ordine:

- ① riempire un vettore **ascisse** con tutti i valori di x
- ② riempire un vettore **ordinate** con i corrispondenti valori di y .
- ③ stampare le coppie ordinate.

	x	y
0	-1.0	0.0
1	-0.5	0.01
.	.	

10	0	1.0
11	0.1	
.		

	ASCISSE	ORDINATE
20	1.0	4.0



Esercizio

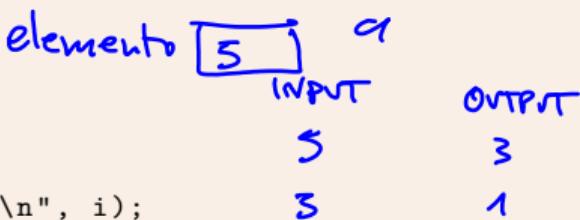
Scrivere un programma che ricerchi in un array di interi un numero richiesto all'utente e, se lo trova, dica a quale indice.

090_array/ricerca-elemento.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 int main() {
5     int a[] = {4, 3, 8, 5, 1};
6     int trovato = 0, i, elemento;
7
8     scanf("%d", &elemento);
9     i = 0;
10    while (!trovato && i < DIM)
11        if (a[i] == elemento)
12            trovato = 1;
13        else
14            i++;
15    if (trovato)
16        printf("Elemento trovato all'indice %d\n", i);
17    return 0;
18 }
```

a[4]	1
a[3]	5
a[2]	8
a[1]	3
a[0]	4



Pattern ricerca

- La ricerca di un elemento di una array è un problema molto comune.
- In generale, si cerca un elemento avente una certa proprietà, ad esempio:
 - avere un certo valore
 - essere positivo $a[i] > 0$
 - non appartenere a un certo insieme
- Nell'esempio alla slide 32 qual è la proprietà desiderata? *ESSERE UGUALE AL NUMERO INSERITO*
- La ricerca si può implementare con un ciclo, con una variabile logica di appoggio *FLAG* (slide 34), o usando l'istruzione **break** per uscire dal ciclo quando si trova l'elemento (slide 35).

Ricerca (con flag)

Ricerca di un elemento che ha una certa proprietà.

090_array/ricerca-flag.c

```

1 #define DIM 10
2 int main() {
3     int a[DIM]; Flag
4     int trovato = 0, i;
5     i=0, a[i]==3
6     while (!trovato && i < DIM) a[i]>0
7         if /* proprietà desiderata di a[i] */
8             trovato = 1;
9         else
10            i++;
11         Flag
12     if (trovato) i contiene l'indice dell'elemento cercato
13     /* fai qualcosa con i o a[i] */
14 }
```

Se la condizione di permanenza fosse solo $i < \text{DIM}$, il programma funzionerebbe?

Ricerca (senza flag)

Ricerca di un elemento che ha una certa proprietà.

090_array/ricerca-break.c

```

1 #define DIM 10
2 int main() {
3     int a[DIM];
4     int i;
5
6     for (i = 0; i < DIM; i++)
7         if /* proprietà desiderata di a[i] */
8             break;
9
10    if (i < DIM)   in i c'è l'indice dell'elemento cercato
11    /* fai qualcosa con i o a[i] */
12 }
```



Problema

^(INPUT)
2, 0, -1, 1, -3

Riempire un array di interi con gli elementi positivi contenuti in un altro array di interi.

Proviamo con il pattern map:

090_array/positivi-map.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM], b[DIM], i, j = 0;
6
7     for (i = 0; i < DIM; i++) {
8         scanf("%d", &a[i]);
9         for (i = 0; i < DIM; i++)
10            if (a[i] > 0)
11                b[i] = a[i];
12 }
13
14     b[i] = a[i] > 0 ? a[i] : b[i];

```

for each
(input, a)
map

a[0]	-3	b[0]	
a[1]	1	b[1]	1
a[2]	-1	b[2]	
a[3]	0	b[3]	
a[4]	2	b[4]	2

b[0]	?
b[1]	?
b[2]	?
b[3]	?
b[4]	?

b[0]	1
b[1]	2

DL [2]

La soluzione è corretta? No

DIMENSIONE
LOGICA 2

Dimensione fisica e logica

- Il problema della soluzione proposta alla slide 36 è che il numero di elementi di **b** non è noto a priori; in generale, sarà compreso fra **0** e il numero di elementi di **a**.
- Un array è una collezione finita di N celle dello stesso tipo; questo non significa che si debbano per forza usare sempre tutte!
- Un array può avere una **dimensione logica** (cioè il numero di elementi significativi) inferiore (e mai superiore!) alla sua **dimensione fisica** (cioè il numero di celle di memoria riservate per l'array quando è stato definito).
- Quindi: se non sappiamo esattamente quanti elementi conterrà un array,
 - scegliamo una dimensione fisica sufficiente **int b[DIM]**
 - rappresentiamo la dimensione logica con una variabile, che aggiorneremo quando necessario.
- I pattern su array (ad esempio ForEach) devono tenere conto della dimensione logica.

for (i=0, i < dl; i++)
printf ("%d", b[i]),

Positivi

090_array/positivi.c

```
1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM], b[DIM], i, dl = 0;
6
7     for (i = 0; i < DIM; i++)
8         scanf("%d", &a[i]);
9     for (i = 0; i < DIM; i++)
10        if (a[i] > 0) {
11            b[dl] = a[i];
12            dl++;
13        }
14 }
```

i 0 1 2 3 4 5

dl 0 1 2

a[0]	-3
a[1]	1
a[2]	-1
a[3]	0
a[4]	2

b[0]	
b[1]	
b[2]	
b[3]	
b[4]	1
b[5]	2

for (i = 0, i < dl, i++)
 printf("%d\n", b[i]),

Pattern Filter filter(f , s_1)

- Il programma alla slide 38 è un esempio del pattern **filter**, che compone un **vettore di destinazione** con gli elementi di un **vettore di origine** che hanno una certa proprietà.
b *a*
70
- La dimensione logica finale del vettore di destinazione non è nota, perché non sappiamo quanti elementi del vettore di origine hanno la proprietà desiderata.
- La dimensione fisica del vettore di destinazione deve essere grande almeno quanto la dimensione logica del vettore di origine.
- Il pattern si implementa (ovviamente) con un ciclo.
Nel corpo del ciclo, l'indice del vettore di origine (che ha **N** elementi) varia da **0** a **N-1**, mentre l'indice del vettore di destinazione viene incrementato solo dopo che vi è stato inserito un elemento.

Pattern Filter

090_array/filter.c

```

1 #include <stdio.h>
2 #define DIM 10
3
4 main() { *ORIGINE *DESTINAZIONE
5     int a[DIM], b[DIM], i, j; di
6     // di ...
7     j = 0;
8     for (i = 0; i < DIM; i++)
9         if (/* proprieta` desiderata di a[i] */) {
10             b[j] = a[i]; INSERISCI a[i] in b
11             di j++;
12         }
13     // qui b ha dimensione logica j e contiene di b[0], ..., b[j-1]
14     // tutti gli elementi di a con la
15     // proprieta` desiderata
16 }
```

Esercizio

Anni bisestili

Scrivere un programma che:

- ① crei un array **a** con gli interi compresi fra 1900 e 2100;
- ② crei un secondo array **b** con gli elementi di **a** che rappresentano anni bisestili
- ③ stampi il contenuto di **b**.

$$a[0] = 1900$$

$$a[1] = 1901$$

:

:

$$b \cdot 1904, 1908, \dots, 2092, 2096$$

$$a[7] = 2100$$

INPUT 2 5 -1 4 2

Somma

Il seguente programma salva **DIM** numeri richiesti all'utente in un array, e poi ne stampa ~~la~~ **la** somma.

090_array/somma-elementi.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM], i, s;
6
7     for (i = 0; i < DIM; i++) {
8         scanf("%d", &a[i]);
9         s = 0; INIZIALIZZAZIONE ACCUMULATORE
10    for (i = 0; i < DIM; i++)
11        s = s + a[i]; AGGIORNAMENTO ACCUMULATORE
12    } OPERAZIONE SOMMA DEGLI ELEMENTI DELL'ARRAY

```

a[4]	2
a[3]	-1
a[2]	3
a[1]	2
a[0]	

s **02584810**

i **072345**

REDUCE (+, 0, a)

Pattern Reduce

Il programma alla slide 42 è un'applicazione del pattern Reduce (o FoldL), che serve a calcolare un valore

- partendo da un **valore iniziale** v_0
- ottenendo v_{i+1} applicando un'**operazione** a v_i e all'elemento i -esimo del **vettore**.

*A VALORE INIZIALE
ACCUMULATORE*

Nel caso alla slide 42, il valore iniziale è 0 e l'operazione è la somma + fra due interi. Si può implementare usando un accumulatore per i valori successivi di v_i , e aggiornandolo con un ciclo **for** che fa variare i su tutti gli indici dell'array.

$$S = v_0$$

for ($i=0$, $i < \text{dim}$; $i++$)

$S = \text{operazione}(s, a[i]),$

Reduce

090_array/reduce.c

```
1 #include <stdio.h>
2 #define DIM 10
3
4 main() {
5     int a[DIM], i, s;
6     // ...
7     s = 0; // inizializzazione accumulatore
8     for (i = 0; i < DIM; i++)
9         s = /* aggiornamento accumulatore
10            al risultato dell'operazione su se
11            stesso (s) e l'elemento corrente (a[i]) */
12 }
```

Esercizio

Prodotto

Scrivere un programma che richieda all'utente 5 numeri, e poi stampi una riga contenente l'uguaglianza fra l'espressione prodotto dei cinque numeri inseriti e il risultato della moltiplicazione. Ad esempio:

$$2 \times 4 \times 5 \times 3 \times 2 = 240$$

(INPUT

2 4 5 3 2

4	2
3	3
2	5
1	4
0	2
	a

for each (input, a)

reduce (*, 1, a)

int p = 1;

p = 2 * 4 * 5 * 3 * 2 = 240

Massimo

Il seguente programma richiede all'utente **DIM** numeri interi e li salva in un array; dopo di che stampa l'indice dell'array contenente l'intero massimo.

090_array/massimo.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 main() {
5     int a[DIM], i, m;
6
7     for (i = 0; i < DIM; i++)
8         scanf("%d", &a[i]);
9     m = 0;
10    for (i = 1; i < DIM; i++)
11        if (a[i] > a[m])
12            m = i;
13    printf("%d\n", m);
14 }
  
```

a[4]	1
a[3]	4
a[2]	3
a[1]	0
a[0]	2

m 023
 i 12345

ACCUMULATORE
 $m = \underline{a[i] > a[m] ? i : m;}$

indice elem massimo

a[m] elem massimo

reduce (, 0, a[1] . a[2])

Ricerca del massimo

In generale, si può voler ricercare l'elemento di un array che massimizza una qualsiasi proprietà (non necessariamente il valore dell'elemento come alla slide 46). Ad esempio, l'elemento

- di valore minimo
- di valore assoluto massimo
- che si trova per primo in un secondo array

int $a[1 = \{ -4, 3, 0 \}]$

Pattern ricerca del massimo

090_array/ricerca-massimo.c

```
1 #include <stdio.h>
2 #define DIM 10
3
4 main() {
5     int a[DIM], i, m;
6
7     // ...
8     m = 0; INIZIALIZZAZIONE
9     for (i = 1; i < DIM; i++)
10        if /* a[i] supera a[m] secondo
11           la proprietà desiderata */
12            m = i; I DIVENTA IL NUOVO INDICE DEL MASSIMO
13
14    // l'elemento massimo è all'indice m
15 }
```

INDICE DELL'ELEMENTO MASSIMO

INIZIALIZZAZIONE

I DIVENTA IL NUOVO INDICE DEL MASSIMO

Sommario

1 Array

2 Liste, sequenze e pattern con array

3 Ordine e ordinamento

Parentesi: scambio del contenuto di due variabili

Come scambiare il contenuto di due variabili? Ad esempio, nel seguente programma, le variabili sono stampate nello stesso ordine in cui sono lette, ma se l'utente digita 2 3, il programma dovrebbe stampare 3 2.

090_array/swap-1.c

```

1 #include <stdio.h>
2
3 int main(void) {
4     int a, b;
5     scanf("%d%d", &a, &b);
6     b = a;
7     a = b;
8     printf("%d %d\n", a, b);
9 }
```

scant("%d%d", &a, &b);
 printf("%d %d\n", a, b),

a 2

b 3

Il programma funziona?

Pattern swap

E' necessario usare una terza variabile per salvare il contenuto della variabile **a** prima che venga sovrascritta con il valore di **b**.

090_array/swap.c

```
1 #include <stdio.h>
2
3 int main(void) {
4     int a, b;
5     scanf ("%d%d", &a, &b);
6
7     {
8         int temp = b;
9         b = a;
10        a = temp;
11    }
12
13    printf ("%d %d\n", a, b);
14 }
```

a 2/3
b 3/2
temp 3

Esercizio

INPUT

2 6 4 0 5

OUTPUT

0 2 4 5 6



Stampa ordinata

Scrivere un programma che richieda all'utente ~~5~~⁵ numeri e li stampi dal più piccolo al più grande.

E' evidente che gli elementi devono essere tutti memorizzati prima che si inizi a stamparli (ad esempio perché fino all'ultimo elemento non sappiamo quale sia il minimo).

Come assicurarsi che siano stampati nell'ordine desiderato?

$a[4]$	5
$a[3]$	0
$a[2]$	4
$a[1]$	6
$a[0]$	2

$a[4]$	6
$a[3]$	5
$a[2]$	4
$a[1]$	2
$a[0]$	0

Inserimento ordinato

a	20	642	64	65	6
	0	1	2	3	4

Si può ottenere un array a ordinato (ad esempio in modo crescente) inserendo via via n elementi

d1 642

- partendo da un array vuoto (di dimensione logica 0)
 - inserendo ogni elemento (incrementando di 1 la dimensione logica) in modo che l'array rimanga ordinato. In particolare, ogni inserimento di un elemento m in un array a di dimensione logica $d1$ si ottiene scalando verso destra tutti gli elementi di a maggiori di m , e infine inserendo m al posto dell'ultimo elemento scalato.
- se $m > a[d1-1]$ si termina, altrimenti si assegna $a[d1] = a[d1-1]$ e si continua
 - se $m > a[d1-2]$, si termina, altrimenti si assegna $a[d1-1] = a[d1-2]$ e si continua
 - e così via: se non si è terminato prima, si arriverà ad assegnare $a[1] = a[0]$
 - infine si assegna m all'ultimo elemento scalato verso destra.

Soluzione

090_array/stampa-ordine.c

```

1 #include <stdio.h>
2 #define DIM 5
3 int main(void) {
4     int m, a[DIM], dl, i, j;
5
6     for (dl = 0; dl < DIM; dl++) {
7         scanf("%d", &m);           dimensione logica
8         j = dl;                  m elementi da inserire
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }                          a è ordinato
15     for (i = 0; i < dl; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
  
```

elementi > m scelto verso destra

Inserimento ordinato

090_array/inserimento-ordinato.c

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int m, a[DIM], dl, j;
6
7     for (dl = 0; dl < DIM; i++) {
8         scanf("%d", &m);
9         j = dl;           m > a[j-1]
10        while (j > 0 && /* m precede a[j-1] secondo il criterio usato */) {
11            a[j] = a[j - 1];
12            j--;
13        }
14        a[j] = m;
15    }
16    // l'array e` ordinato secondo il criterio usato
17 }
```

Ordinamento

a 2 6 4 0 8
b 0 2 4 5 6

- Capita spesso di avere un array non ordinato e di volerlo rendere ordinato.
- Questo problema è detto di **ordinamento**.
- Conoscendo la tecnica di inserimento ordinato, dato un array **a** non ordinato possiamo ordinarlo in due passi:
 - ① creando un array **b** ordinato inserendo in **b**, in modo ordinato, via via tutti gli elementi di **a**;
 - ② copiando **b** in **a**.

Questa tecnica ha però l'inconveniente di richiedere memoria aggiuntiva (per l'array **b**).

- Esistono svariati **algoritmi di ordinamento** che operano *in place*, cioè spostando gli elementi dell'array iniziale senza richiedere altri array di appoggio.
- Noi vedremo i due più semplici: **insertion sort** e **selection sort**. Nella pratica, sono poco usati perché ne esistono di più efficienti.

Relazioni di ordinamento

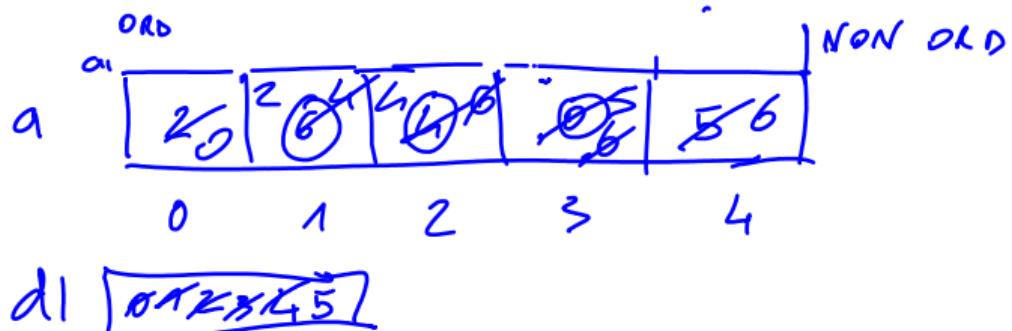
- Nelle prossime diapositive, per semplicità, supporremo di voler ordinare gli elementi secondo la consueta relazione di \leq .
- Le tecniche presentate però funzionano per qualsiasi relazione \leq di ordine totale definita sul tipo di elementi dell'array, cioè una relazione
 - riflessiva ($\forall a(a \leq a)$)
 - antisimmetrica ($\forall a, b(a \leq b \wedge b \leq a \rightarrow a \sim b)$ dove \sim è una relazione di equivalenza)
 - transitiva ($\forall a, b, c(a \leq b \wedge b \leq c \rightarrow a \leq c)$)
 - definita per ogni coppia di valori ($\forall a, b(a \leq b \vee b \leq a)$)
- Ad esempio: $a \leq b \leftrightarrow a^2 \geq b^2$ con $a \sim b \leftrightarrow a^2 = b^2$
- Basta cambiare l'espressione di confronto fra elementi.

Insertion sort

Uno degli algoritmi di ordinamento (detto **insertion sort**) usa la tecnica di inserimento ordinato, ma senza richiedere un secondo array.

E' possibile ordinare un array **a** di dimensione **DIM** in un numero di passi pari a **DIM**, numerati da **0** a **DIM-1**. Al passo **d_l**-esimo, **a** è suddiviso in una **parte ordinata** di dimensione logica **d_l** (indici da **0** a **d_l-1**) e in una **parte non ordinata** di dimensione logica **DIM - d_l** (indici da **d_l** a **DIM - 1**).

- al primo passo, la parte ordinata è vuota;
- ad ogni passo, si inserisce in modo ordinato nella parte ordinata il primo elemento della parte non ordinata.



Insertion sort

090_array/insertion-sort.c

```

1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl]; → PRIMO ELEMENTO PARTE NON ORD
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

$a \boxed{4 \ 1 \ 8 \ 0 \ 5}$

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
	5
a[3]	4294952536
	0
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	4
i	4294952556
	1448436411
dl	4294952552
	0

main (7)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

Diagram illustrating the state of variables during the execution of the insertion sort algorithm.

The array a is shown as a stack of five cells:

$a[4]$	4294952540
	5
$a[3]$	4294952536
	0
$a[2]$	4294952532
	8
$a[1]$	4294952528
	1
$a[0]$	4294952524
	4

Below the array, the current values of i , dl , j , and m are listed:

- i : 1 (highlighted with a red box)
- dl : 0
- j : -14556 (highlighted with a red box)
- m : 1 (highlighted with a red box)

A vertical red line on the right side of the array indicates the boundary of the sorted portion of the array.

main (8)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	4294952536
a[2]	4294952532
a[1]	4294952528
a[0]	4294952524
dl	4294952556
j	448436411
m	4294952548

Non end

main (9)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
dl	1
j	4294952528
m	1
	4294952524
	4
	4294952556
	1
	448436411
	4294952552
	0
	4294952548
	0
	4294952544
	4

main (13)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
NON ORD	
i	4294952528
dl	1
j	1
m	4294952524
	4
ORD	
i	4294952556
dl	1448436411
j	4294952552
m	4294952548
	0
	4294952544
	4

main (8)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
dl	1
j	4294952528
m	1

main (9)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
dl	1
j	4294952528
m	1

main (10)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
dl	4294952528
j	4
m	4294952524

main (11)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
dl	4
j	4294952528
m	4
i	4294952524
dl	4
j	4294952556
m	1448436411

main (13)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
dl	8
j	4294952528
m	4
i	1
	4294952524
	1
	4294952556
	1448436411
	2
	4294952552
	2
	4294952548
	0
	4294952544
	1

Non-ord ord main (8)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540 5
a[3]	4294952536 0
a[2]	4294952532 8
a[1]	4294952528 4
a[0]	4294952524 1
i	4294952556 1
dl	4294952552 2
j	4294952548 2
m	4294952544 8

main (9)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
dl	4
j	1
m	4294952528
	4294952524
	1
	4294952556
	1448436411
	4294952552
	2
	4294952548
	2
	4294952544
	8

main (13)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
<u>NOORD</u>	4294952536
OKD	0
a[2]	4294952532
a[1]	8
a[0]	4294952528
i	4
dl	1
j	4294952556
m	1448436411
	4294952552
	3
	4294952548
	2
	4294952544
	8

main (8)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
dl	4
j	1
m	4294952528
	4294952524
	1
	4294952556
	1
	448436411
	4294952552
	3
	4294952548
	3
	4294952544
	0

main (9)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
dl	4
j	1
m	4294952528
	4294952524
	1
	4294952556
	1
	448436411
	4294952552
	3
	4294952548
	3
	4294952544
	0

main (10)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540 5
a[3]	4294952536 8
a[2]	4294952532 8
a[1]	4294952528 4
a[0]	4294952524 1
i	4294952556 1
dl	4294952552 3
j	4294952548 3
m	4294952544 0

main (11)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
dl	1
j	2
m	0

main (10)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540 5
a[3]	4294952536 8
a[2]	4294952532 4
a[1]	4294952528 4
a[0]	4294952524 1
i	4294952556 1
dl	4294952552 3
j	4294952548 2
m	4294952544 0

main (11)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
dl	1
j	4294952528
m	1

main (10)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
dl	1
j	2
m	3

main (11)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540 5
a[3]	4294952536 8
a[2]	4294952532 4
a[1]	4294952528 1
a[0]	4294952524 1
i	4294952556 1
dl	4294952552 3
j	4294952548 0
m	4294952544 0

main (13)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
dl	1
j	2
m	3

main (8)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
dl	1
j	4294952528
m	1
	0
	4294952524
	0
	4294952556
	1
	448436411
	4294952552
	4
	4294952548
	4
	4294952544
	5

main (9)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
dl	1
j	2
m	3

main (10)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540 8
a[3]	4294952536 8
a[2]	4294952532 4
a[1]	4294952528 1
a[0]	4294952524 0
i	4294952556 1448436411
dl	4294952552 4
j	4294952548 4
m	4294952544 5

main (11)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

a[4]	4294952540 8
a[3]	4294952536 8
a[2]	4294952532 4
a[1]	4294952528 1
a[0]	4294952524 0
i	4294952556 1448436411
dl	4294952552 4
j	4294952548 3
m	4294952544 5

main (13)

Insertion sort

```
1 #include <stdio.h>
2 #define DIM 5
3
4 int main(void) {
5     int a[] = {4, 1, 8, 0, 5}, i, dl = 0;
6
7     for (dl = 0; dl < DIM; dl++) {
8         int j = dl, m = a[dl];
9         while (j > 0 && m < a[j - 1]) {
10             a[j] = a[j - 1];
11             j--;
12         }
13         a[j] = m;
14     }
15     for (i = 0; i < DIM; i++)
16         printf("%d ", a[i]);
17     printf("\n");
18 }
```

For Each (stampa, a)

a[4]	4294952540 8
a[3]	4294952536 5
a[2]	4294952532 4
a[1]	4294952528 1
a[0]	4294952524 0
i	4294952556 1448436411
dl	4294952552 5

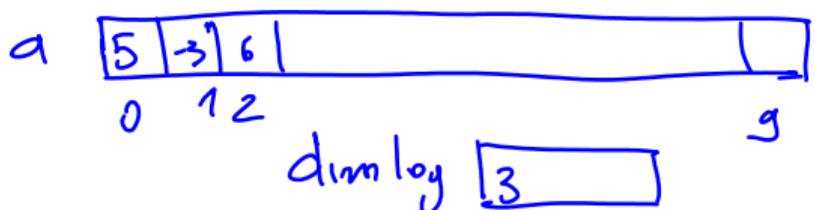
main (15)

Esercizio

Insertion sort per quadrato

Scrivere un programma che

- ① richieda in ingresso un massimo di 10 numeri interi non nulli (fermandosi quando l'utente digita 0) memorizzandoli in un array;
- ② ordini l'array secondo l'ordine crescente dei quadrati dei suoi elementi, usando l'algoritmo insertion sort;
- ③ stampi l'array.



Selection sort

L'idea dell'algoritmo **selection sort** è

- il primo elemento dell'array ordinato è il minimo dell'array iniziale
- il secondo elemento è il minimo dell'array iniziale, tolto il primo elemento dell'array ordinato
- ...
- l'ultimo elemento dell'array ordinato è l'ultimo rimasto

Un modo per implementarlo è con un ciclo in cui la variabile **i** di controllo varia fra **0** e **DIM - 2**. All'iterazione **i**-esima, se l'elemento di indice **i** è maggiore del minimo della parte di array di indici da **i+1** a **DIM - 1**, scambiarli.

a	$\boxed{10 \mid 1 \mid 8 \downarrow \mid 5^8 \mid 5^8}$
	$\underline{0 \quad 1 \quad 2 \quad 3 \downarrow \quad 4}$

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5     indice del minimo
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
	4294952528
	1
	4294952524
	4
	4294952556
	1448436427

main (6)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	0
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	4
i	4294952556
	0..
min	4294952552
	0
j	4294952548
	-14556

main (8)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
min	4294952528
j	1

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	0
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	4
i	4294952556
	0
min	4294952552
	0
j	4294952548
	1

main (10)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	0
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	4
i	4294952556
	0
min	4294952552
	1
j	4294952548
	2

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	0
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	4
i	4294952556
	0
min	4294952552
	1
j	4294952548
	3

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	0
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	4
i	4294952556
	0
min	4294952552
	1 3
j	4294952548
	3

main (10)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	0
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	4
i	4294952556
	0
min	4294952552
	3
j	4294952548
	4

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

SWAP

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	8
min	1
j	4294952528
	4
	4294952556
	0
	4294952552
	3
	4294952548
	5

main (11)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	0
a[0]	4294952532
i	1
min	4294952528
j	4
temp	4294952524

main (12)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540 5
a[3]	4294952536 04
a[2]	4294952532 8
a[1]	4294952528 1
a[0]	4294952524 4
i	4294952556 0
min	4294952552 3
j	4294952548 5
temp	4294952544 0

main (13)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540 5
a[3]	4294952536 4
a[2]	4294952532 8
a[1]	4294952528 1
a[0]	4294952524 40
i	4294952556 0
min	4294952552 3
j	4294952548 5
temp	4294952544 0

main (14)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

main (8)

a[4]	4294952540 5
a[3]	4294952536 4
a[2]	4294952532 8
a[1]	4294952528 1
a[0]	4294952524 0
i	4294952556 1
min	4294952552 1
j	4294952548 5

Non ord ord

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	1
min	4294952552
	1
j	4294952548
	2

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	1
min	4294952552
	1
j	4294952548
	3

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	1
min	4294952552
	1
j	4294952548
	4

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	1
min	4294952552
	1
j	4294952548
	5

main (11)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

main (8)

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	4
a[0]	4294952532
i	8
min	1
j	2

NON ORD

ORD

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	2
min	4294952552
	2
j	4294952548
	3

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	2
min	4294952552
	2
j	4294952548
	3

main (10)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	2
min	4294952552
	3
j	4294952548
	4

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

main (11)

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	4294952532
a[0]	8
i	4294952528
min	1
j	0

The table shows the state of the array a and variables i, min, and j during the execution of the selection sort algorithm. The array a contains the values [4, 1, 8, 0, 5]. The variable i is highlighted with a red box at index 11. The variable min is circled in blue at index 11. The variable j is circled in red at index 16.

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	2
min	4294952552
	3
j	4294952548
	5
temp	4294952544
	0

main (12)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
	5
a[3]	4294952536
	4
a[2]	4294952532
	8
a[1]	4294952528
	1
a[0]	4294952524
	0
i	4294952556
	2
min	4294952552
	3
j	4294952548
	5
temp	4294952544
	4

main (13)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540 5
a[3]	4294952536 8
a[2]	4294952532 8
a[1]	4294952528 1
a[0]	4294952524 0
i	4294952556 2
min	4294952552 3
j	4294952548 5
temp	4294952544 4

main (14)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
<u>a[2]</u>	8
a[1]	4294952532
a[0]	4
i	1
min	0
j	4294952528
	1
	4294952524
	0
	4294952556
	3
	4294952552
	3
	4294952548
	5

main (8)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
min	1
j	0

main (9)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
min	1
j	2

main (10)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540 5
a[3]	4294952536 8
a[2]	4294952532 4
a[1]	4294952528 1
a[0]	4294952524 0
i	4294952556 3
min	4294952552 4
j	4294952548 5

main (11)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
min	1
j	2
temp	0
	4294952528
	4294952524
	4294952556
	4294952552
	4294952548
	4294952544

main (12)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

a[4]	4294952540
a[3]	5
a[2]	4294952536
a[1]	8
a[0]	4294952532
i	4
min	1
j	2
temp	0

main (13)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

Non ORD → ORD

	a[4]	4294952540 8
	a[3]	4294952536 85
	a[2]	4294952532 4
	a[1]	4294952528 1
	a[0]	4294952524 0
i		4294952556 3
min		4294952552 4
j		4294952548 5
temp		4294952544 5

main (14)

Selection sort

```
1 #include <stdio.h>
2 #define DIM 5
3 int main() {
4     int a[] = {4, 1, 8, 0, 5}, i;
5
6     for (i = 0; i < DIM - 1; i++) {
7         int min = i, j;
8         for (j = i + 1; j < DIM; j++)
9             if (a[j] < a[min])
10                 min = j;
11         if (i != min) {
12             int temp = a[min];
13             a[min] = a[i];
14             a[i] = temp;
15         }
16     }
17     for (i = 0; i < DIM; i++)
18         printf("%d ", a[i]);
19     printf("\n");
20 }
```

The diagram illustrates the state of the array `a` and the variable `i` at the start of line 17. A blue bracket on the left groups the first five rows of the table, corresponding to the elements of `a` from index 0 to 4. To the right of the table, a vertical red line is labeled `main (17)`, indicating the current line of execution. The table shows the following values:

a[4]	4294952540
a[3]	8
a[2]	4294952536
a[1]	5
a[0]	4294952532
i	4
	4294952528
	1
	4294952524
	0
	4294952556
	4

Esercizio

float a[10];

Selection sort per coseno decrescente

Scrivere un programma che

- ① richieda in ingresso 10 numeri reali memorizzandoli in un array;
- ② ordini l'array secondo l'ordine decrescente dei coseni dei suoi elementi, usando l'algoritmo di selection sort;
- ③ stampi l'array.

include <math.h>

Usare la funzione `double cos(double x)` dichiarata in `math.h`.

$$\cos(a[j]) > \cos(a[min])$$