PASCAL: I VETTORI

TRATTO DA CAMAGNI-NIKOLASSY, CORSO DI INFORMATICA, VOL. 1, HOEPLI

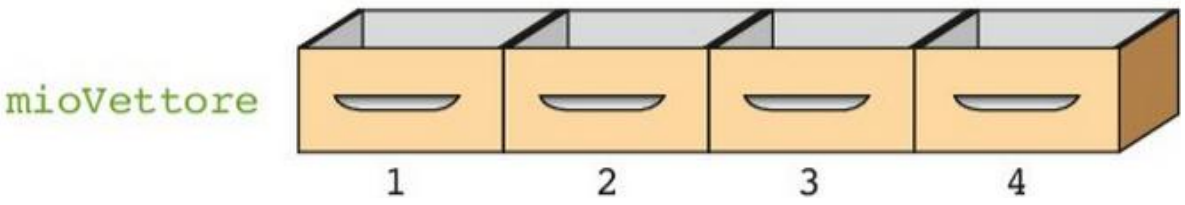
Informatica

I dati strutturati:

gli array

I vettori (o Array)

- Fino a ora abbiamo memorizzato le informazioni (valori) in una variabile
- Una variabile può contenere un solo dato
- Per poter memorizzare più informazioni in una sola variabile abbiamo bisogno di un *Vettore*



Un vettore (o Array) è l'insieme di più variabili dello stesso tipo "affiancate" tra loro

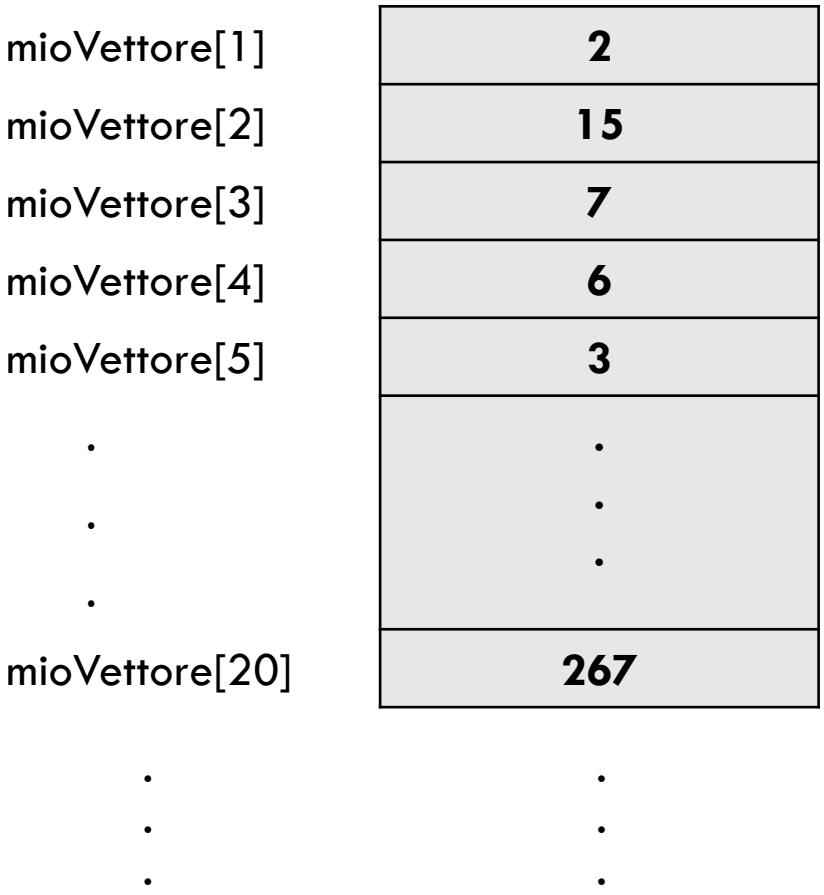
Schematizzazione logica

- La memoria centrale viene schematizzata come una striscia continua di locazioni di memoria, numerate in ordine crescente

Cella 0	**23**
Cella 1	**C**
Cella 2	**I**
Cella 3	**A**
Cella 4	**O**
Cella 5	**65,789**
Cella 6	
.	.
.	.
.	.

Schematizzazione logica

- Un array può essere visto come uno spazio contiguo di celle di memoria



Dichiarazione di un Array

- Per poter utilizzare un vettore è necessario definire tre elementi che lo caratterizzano:
  - **L'identificatore** del vettore (nome)
  - Il **numero di elementi** che lo costituiscono
  - Il **tipo di elementi** che deve contenere

- In linguaggio Pascal si ha:

```
var
  mioVettore:array[1..20]of integer;
```

L'istruzione si compone dei seguenti elementi:

- `mioVettore` : identificatore (nome) della variabile vettore;
- `1..20` : dimensione, cioè inizio e fine del vettore;
- `integer` : tipo di dati contenuti (`integer`, `real`, `char`, `boolean`).

Dichiarazione di un Array: struttura sintattica

- La struttura sintattica è:


```
var
  mioVettore:array[1..20]of integer;
```

Scrittura e lettura di un Array

- Ogni elemento del vettore prende il nome di **cella**
- Ogni cella viene univocamente individuato mediante la sua *posizione*, che è un *numero intero*, chiamato **indice**
- L'indice è espresso nelle [ ]
- L'indice è compreso tra due valori, *inferiore* e *superiore*, definito dal programmatore

Per esempio, se consideriamo un vettore di 20 celle, il primo e l'ultimo elemento saranno:

```
mioVettore[1]
.
.
.
mioVettore[20]
```

Elemento di posizione 1 della variabile ***mioVettore***

Elemento di posizione 20 della variabile ***mioVettore***

Scrittura di un Array (1)

- Per assegnare un valore ad un elemento del vettore è sufficiente individuare la posizione della cella mediante il giusto valore di indice e utilizzare l'operatore di assegnazione :=

```
mioVettore[1]:=123;          // scrittura nel primo elemento
mioVettore[4]:=12;           // scrittura nel quarto elemento
mioVettore[20]:=452;         // scrittura nell'ultimo elemento
```

Scrittura di un Array (2): ciclo a conteggio

- Si utilizza l'indice di conteggio del **ciclo *for*** come indice del vettore
  - ***Es.: Inizializzazione***

```
for indice:=1 to 20 do
  mioVettore[indice]:=0;          // azzeramento di tutte le celle del vettore
```

  - ***Es.: Inserimento da tastiera***

```
for indice:=1 to 20 do
  readln(mioVettore[indice]); // memorizza elemento di posizione indice
```

Equivale a scrivere:

```
readln(mioVettore[1]);                  // memorizza elemento di posizione 1
readln(mioVettore[2]);                  // memorizza elemento di posizione 2
readln(mioVettore[3]);                  // memorizza elemento di posizione 3
. . .
readln(mioVettore[20]);                 // memorizza elemento di posizione 20
```

Scrittura di un Array (3): esercizio

- Definisci un Array di 20 elementi interi
- Genera 20 numeri casuali con la funzione RANDOM
- Memorizza i valori generati nell'array.

Lettura da un Array

- La lettura di un elemento dell'array viene effettuata allo stesso modo di una normale variabile, specificando però l'indice della cella
- Una volta che si specifica l'indice, l'elemento di un array si comporta come una normale variabile

Con lettura di un Array si intende il prelevamento del valore memorizzato nelle celle dell'Array stesso.

Visualizzazione al contrario

- Supponiamo di voler visualizzare alla rovescia sul monitor il contenuto di un array di 20 elementi

```
for conta:=1 to 20 do
  writeln(mioVettore[21-conta]); // visualizza un elemento
```

conta	21- conta	mioVettore(21- conta)
1	21-1 =20	mioVettore(20)
2	21-2 =19	mioVettore(19)
3	21-3 =18	mioVettore(18)
.	.	.
20	21-20 = 1	mioVettore(1)

Esempio

- Leggi da tastiera 20 numeri e memorizzali in un Array.
  - *Esempio1*: Stampa a monitor i valori memorizzati in ordine crescente dell'indice
  - *Esempio 2*: Stampa a monitor i valori memorizzati in ordine invertito (ordine decrescente dell'indice)

Soluzione esempio 1

```
0  program ribalta;
1  (* ribaltamento di un vettore di 20 elementi *)
2  var
3    conta,indice:integer;                  // indici dei cicli a conteggio
4    mioVettore:array[1..20]of integer;
5  begin
6    writeln('inserisci 20 numeri ed io te li ribalto!');
7  (* lettura dei dati e memorizzazione in un vettore *)
8    for indice:=1 to 20 do
9      read(mioVettore[indice]);
10 (* visualizzazione dei dati in ordine rovesciato *)
11   for conta:=1 to 20 do
12     write(' ',mioVettore[   conta]);
13 end.
```

Soluzione esempio 2

```
0  program ribalta;
1  (* ribaltamento di un vettore di 20 elementi *)
2  var
3    conta,indice:integer;                  // indici dei cicli a conteggio
4    mioVettore:array[1..20]of integer;
5  begin
6    writeln('inserisci 20 numeri ed io te li ribalto!');
7  (* lettura dei dati e memorizzazione in un vettore *)
8    for indice:=1 to 20 do
9      read(mioVettore[indice]);
10 (* visualizzazione dei dati in ordine rovesciato *)
11   for conta:=1 to 20 do
12     write(' ',mioVettore[21-conta]);
13 end.
```

Le costanti

- Una **costante** è una cella di memoria (del tutto simile a quella che contiene una variabile) nella quale viene inserito un valore che non può essere modificato in tutto il programma.
- Non può essere modifica durante l'esecuzione del programma (rimane "costante")

Come si definisce in Pascal una costante?

```
program <nomeprogramma>
const                                   // elenco delle costanti
var                                     // elenco delle variabili
begin                                   // corpo del programma
...
end.
```

Definizione parametrica di un Array

- Possiamo definire il numero delle celle di un Array (lunghezza dell'Array) mediante una costante

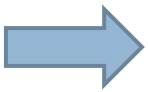
```
const
  dim=30;                    // costante che indica la dimensione del problema
var
  mioVettore:array[1..dim]of integer;          // vettore di dim elementi
for indice:=1 to dim do
  read(mioVettore[indice]);                    // legge dim elementi
for conta:=1 to dim do
  write(' ',mioVettore[dim+1-conta]);          // scrive dim elementi
```

Da notare che dim viene usato anche come estremo del *for*

Vettori con indici particolari

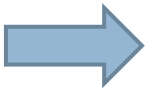
- Possiamo definire Array il cui primo elemento può avere un indice diverso da 1

```
mioVettore:array[0..20]of integer;
```



Array di 21 elementi (da 0 a 20)

```
mioVettore:array[-2..2]of integer;
```



Array di 5 elementi (da -2 a 2)

Ricorda di porre particolare attenzione al valore degli indici di un vettore, sia in fase di scrittura che di lettura.
Nel caso in cui si indicizzi un elemento non presente nel vettore alcuni compilatori Pascal segnalano l'errore con il messaggio "Out of array bound", cioè "Fuori dai margini dell'array" e il programma si sospende; altri compilatori invece, tra cui il Dev-Pascal, non danno alcuna segnalazione e leggono "erroneamente" lo stesso dei valori non significativi che sono presenti all'interno della memoria RAM, riportando risultati indesiderati e non voluti, all'insaputa del programmatore.
Nei nostri programmi questo tipo di errore è di difficile individuazione e quindi ti consigliamo di prestare particolare attenzione a "non uscire dai margini", ricontrollando sempre il valore degli indici e confrontandolo con quello dei due estremi del vettore.

Riepilogo

ABBIAMO IMPARATO CHE...

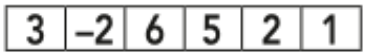
- Per poter memorizzare più di un valore nella stessa variabile si utilizzano i vettori (array).
- Una array è composto da un insieme di celle dello stesso tipo (omogenee) individuate mediante la posizione che assumono nella struttura: prende il nome di indice.
- La dichiarazione di un vettore è l'operazione che permette di definire il numero di elementi che lo compongono: in particolare si definisce il nome, la dimensione e il tipo di elementi che deve contenere.
- È possibile anche definire vettori con estremi negativi
- Dato che sappiamo a priori il numeri di elementi, la figura di controllo più indicata per manipolare i vettori è il ciclo a conteggio.

Esercizi 1.1-1.4, pag. 387

1 Di quanti elementi è composto il seguente vettore?

```
mioVettore: array[-5..0]of integer
```

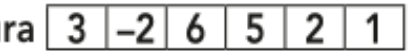
a. –5 b. 5 c. 6 d. 50

2 In riferimento al vettore `vet: array[0..5]` di figura 

indica il risultato della seguente istruzione:

```
writeln(vet[1]-vet[4]+vet[2]);
```

a. –4 b. 2 c. 4 d. –6

3 In riferimento al vettore vet: `array [1..6]` di figura 

indica il risultato della seguente istruzione:

```
writeln(vet[1]-vet[4]+vet[2]);
```

a. –5 b. 7 c. –4 d. 4

4 In riferimento al vettore dell'esercizio 3 indica il risultato della seguente istruzione:

```
for x:=1 to 6 step 2 do
  tot := tot+vet[x]
```

a. 4 b. 11 c. 13 d. 15

Esercizi 2.1-2.2, pag. 387

1. Che risultato produce il codice seguente?

```
var vet1: array[1..8]of integer;
begin
  for x:=1 to 6 do
    begin
      vet1[x]:=x-2;
      vet1[x+1]:=vet1[x];
    end;
  for x:=1 to 6
    write(vet1[x]);
end.
```

2 Che risultato produce il codice seguente?

```
var vet1: array[1..8]of integer;
begin
  for x:=1 to 6 do
    begin
      vet1[x]:=x*2;
      vet1[x+1]:=vet1[x]+x;
    end;
  for x:=1 to 6 do
    write(vet1[x]);
end.
```

Esercizi 2.3-2.4, pag. 387

3 Che risultato produce il codice seguente?

```
var vet1[1..8]of integer;
begin
  for x:=1 to 6 do
    begin
        vet1[x]:=(x-1)*2;
        vet1[x+1]:=vet1[x];
      end;
    for x:=1 to 6 do
      write(vet1[x]);
  end.
```

4 Che risultato produce il codice seguente?

```
begin
  for x:=1 to 6 do
    begin
      vet1[x]:=(x-2)*2;
      vet1[x+1]:=vet1[x];
    end;
  for x:=2 to 6 do
    vet1[1]:=vet1[x]+vet1[1];
  writeln(vet1[1]);
end.
```

Esercizi 2.5, pag. 387

5 Che risultato produce il codice seguente?

```
begin
  for x:=1 to 6 do
    begin
      vet1[x]:=(x-1)*2;
      vet1[x+1]:=vet1[x];
    end;
  for x:=6 downto 2 do
    vet1[1]:=vet1[1]+vet1[x];
  writeln(vet1[1]);
end.
```

Esercizi 1-8, pag. 388

1. Scrivi un programma che genera casualmente *N* numeri, li memorizza in un vettore e visualizza il loro quadrato e la loro somma.
2. Scrivi un programma che calcola la media dei voti della tua pagella dopo averli memorizzati in un vettore di tipo real.
3. Scrivi un programma che genera casualmente 20 numeri, li memorizza in un array e calcola la somma degli elementi di posizione pari e di quelli di posizione dispari.
4. Scrivi un programma che genera casualmente 20 numeri, li memorizza in due array, uno che contiene i numeri pari e l'altro che contiene i numeri dispari, e quindi visualizza i due array sullo schermo.
5. Scrivi un programma che legge N numeri e dopo averli memorizzati in un vettore ribalta il contenuto del vettore in un secondo vettore.
6. Scrivi un programma che legge N numeri e dopo averli memorizzati in un vettore ribalta il contenuto del vettore nello stesso vettore.
7. Scrivi un programma che legge 20 numeri di valore compreso tra 1 e 999 e li memorizza in tre vettori: rispettivamente nel **`vet1`** inserisce i numeri con valore minore di 10, nel **`vet2`** inserisce i numeri con valore minore di 100 e memorizza i rimanenti nel terzo vettore.
8. Scrivi un programma che simula il gioco dei dadi: estrae un numero compreso tra 1 e 6 e conta la frequenza di ciascun numero in una settimana di gioco (5000 esecuzioni).

   Traccia per la soluzione: nella cella **`lanci[num]`** memorizza il numero di volte che un numero num viene estratto, cioè a ogni estrazione viene incrementato di 1 il valore della cella a cui corrisponde il contatore del numero.

Esercizi 9-14, pag. 388

9 Scrivi un programma che simula il gioco della roulette: estrae un numero compreso tra 0 e 90 e conta la frequenza di ciascun numero in una settimana di gioco (5000 esecuzioni).

Traccia per la soluzione: come per l'esercizio precedente, ma con 0 come indice inferiore del vettore.

10 Scrivi un programma che esegue il caricamento random(40) di due vettori di numeri interi; quindi in un terzo vettore effettua la somma degli elementi di posizione corrispondente nei primi due vettori, cioè `vet3[5]:=vet1[5]+vet2[5]`.

Individua quale cella del **vet3** contiene il numero maggiore.

11 Scrivi un programma che dopo aver caricato un vettore di NUM interi, con NUM ≤ 100, generati casualmente con range [10-30], calcola la somma degli elementi pari e la somma degli elementi dispari visualizzando il risultato sullo schermo.

12 Riordina gli elementi di un vettore di *N* numeri interi generati casualmente con range (–20, +20) in modo che tutti i valori negativi stiano nella parte sinistra del vettore e tutti i valori non negativi stiano nella parte destra del vettore, senza utilizzare un array di supporto.

13 Si ricevono come dati d'ingresso due valori interi positivi minori di 100. Memorizza in due array le rappresentazioni dei numeri in base 2. Effettua poi la somma dei due numeri mediante la rappresentazione binaria e memorizza il risultato (sempre rappresentato in binario) in un terzo array.

14 Genera casualmente N numeri, scegli casualmente un elemento del vettore e quindi effettua la *partizione* dell'array rispetto a quell'elemento (cioè elabora il vettore in modo tale che tutti gli elementi a sinistra dell'elemento scelto siano minori o, al limite, uguali, mentre tutti gli elementi a destra siano sempre maggiori).

Ricerca in un vettore

Ricerca in un vettore

- Un problema ricorrente è quello di ritrovare i dati che sono stati precedentemente memorizzati.
- Esistono due tipologie di ricerca:
  - Della **posizione**: cioè conoscendo il valore si vuole verificare se è presente nel vettore e in quale cella è memorizzato
  - Di un **elemento** particolare, ad esempio:
    - Il minimo valore contenuto nel vettore
    - Il massimo valore contenuto nel vettore
    - Gli elementi doppi (ripetuti)
    - Gli elementi di valore dispari
    - Ecc…

Tecniche di ricerca

- Esistono diverse tecniche di ricerca, le più utilizzate sono:
  - Ricerca sequenziale
  - Ricerca con sentinella

Ricerca sequenziale

- Consiste nello “scorrere” tutto il vettore e nell’analizzare il contenuto di tutte le celle.
- Nella slide successiva è riportato un esempio in cui si assegnano valori casuali a un vettore e poi si cerca un determinato valore.

```
0  program ricerca;
1  (* ricerca di un numero in un vettore *)
2  var
3    cerca,indice,posizione:integer;
4    mioVettore:array[1..20]of integer;
5  begin
6    posizione:=0;
7    for indice:=1 to 20 do
8      mioVettore[indice]:=random(100);
9    writeln('inserisci il numero da cercare < 100');
10   read(cerca);
11 (* ricerca sequenziale di un dato *)
12   for indice:=1 to 20 do
13     if mioVettore[indice]=cerca
14       then
15         posizione:=indice;
16   if(posizione>0)
17     then
18       writeln('il numero e'' in posizione ',posizione)
19     else
20       writeln('il numero non e'' presente');
21 end.
```

Esercizio

- Ripeti la ricerca segnalando se il numero ricercato è presente due volte nel vettore.

Ricerca con sentinella

- Nell'esempio precedente abbiamo continuato a cercare anche dopo aver trovato il valore richiesto
- Con la ricerca sequenziale con sentinella si utilizza una variabile booleana nella condizione di ingresso/uscita di un ciclo iterativo. Tale variabile assume il valore "true" quando il valore viene trovato provoca l'uscita dal ciclo e quindi l'interruzione della ricerca.
- Nella slide successiva è riportato un esempi di ricerca con sentinella

```
0  program ricerca2;
1  (* ricerca di un numero in un vettore *)
2  var
3    cerca,indice,posizione:integer;
4    trovato:boolean;                                  // sentinella
5    mioVettore:array[1..20]of integer;
6  begin
7    trovato:=false;
8    posizione:=0;
9    for indice:=1 to 20 do
10     mioVettore[indice]:=random(100);
11   writeln('inserisci il numero da cercare < 100');
12   read(cerca);
13 (* ricerca sequenziale con sentinella *)
14   indice:=1;                                        // inizializza indice di ricerca
15   while(trovato=false)and(indice<=20)do
16     begin
17       if(mioVettore[indice]=cerca)
18         then
19           begin
20             posizione:=indice;
21             trovato:=true;
22           end;
23       indice:=indice+1;                             // incrementa posizione
24     end;
25   if(posizione>0)
26     then
27       writeln('il numero e'' in posizione ',posizione)
28     else
29       writeln('il numero non e'' presente');
30 end.
```

Ricerca con sentinella (2)

- Supponiamo che nel vettore siano memorizzati dei numeri in ordine crescente
- In tal caso se il numero cercato è più piccolo del valore corrente presente nel vettore allora si può interrompere la ricerca
- Bisogna semplicemente modificare l'istruzione 15:

```
while(trovato=false)and(indice<=20)or(mioVettore[indice]>cerca) do
```

Riepilogo

- Un problema fondamentale nell'informatica è quello di ritrovare i dati che sono stati precedentemente memorizzati.
- Esistono due tipologie di ricerca:
  1. ricerca della posizione di un particolare elemento;
  2. ricerca di un elemento particolare.
- L'algoritmo di ricerca seriale (o sequenziale) consiste nello scorrere tutte le posizione del vettore fino a che non si trova corrispondenza con il valore cercato, che potrebbe anche non essere presente nel vettore.
- Viene utilizzata una variabile booleana come sentinella che per mette di interrompere il ciclo di scansione quando l'elemento cercato viene individuato.

Esercizi 1-7, pag. 395

1 Scrivi un programma che genera casualmente 30 numeri di valore minore di 1000, li memorizza in un vettore e visualizza il più piccolo e il più grande.

2 Scrivi un programma che genera casualmente *N* numeri di valore minore di 1000 (dove *N* è una costante manifesta), li memorizza in un vettore e cancella i numeri pari presenti sostituendoli con 0. Visualizza quindi il vettore.

3 Scrivi un programma che genera casualmente *N* numeri di valore minore di 1000 (dove *N* è una costante manifesta), li memorizza in un vettore e cancella i numeri inferiori di un numero MAX letto da tastiera. Visualizza quindi il vettore.

4 Scrivi un programma che genera casualmente *N* numeri di valore minore di 1000 (dove *N* è una costante manifesta), li memorizza in un vettore e quindi separa i numeri pari dai numeri dispari copiandoli rispettivamente in due vettori. Visualizza quindi i due vettori senza visualizzare i campi di valore 0.

5 Scrivi un programma che genera casualmente *N* numeri di valore minore di 1000 (dove *N* è una costante manifesta), li memorizza in un vettore e ricerca la posizione di tutti gli elementi che hanno valore uguale a un numero `NUM` inserito da tastiera. Visualizza quindi il vettore.

6 Scrivi un programma che genera casualmente *N* numeri di valore minore di 1000 (dove *N* è una costante manifesta), li memorizza in un vettore ed effettua lo scambio di due numeri in posizione adiacente se il primo numero è maggiore del secondo, cioè se vet[x]>vet[x+1]. Al termine della elaborazione visualizza il vettore risultante: che particolare valore ha l'ultimo elemento del vettore?

7 Nell'array `Ore_Studio` sono memorizzate le ore passate a studiare da uno studente per ogni giorno del mese (generate casualmente con valori 0-4). Calcola e visualizza il numero totale di ore passate a studiare nel corso del mese, il giorno (o i giorni) in cui lo studente ha studiato per un numero maggiore di ore e il numero di giorni in cui non ha aperto libro.

Esercizi 8-13, pag. 395

8 In un array di 12 elementi sono memorizzate le presenze mensili di un albergo nel corso dell'anno. Calcola:
- la media di presenze nel corso dell'intero anno;
- il numero totale di presenze nei mesi estivi (6, 7, 8);
- il mese in cui si è registrato il numero massimo di presenze;
- il mese in cui si è registrato il numero minimo;
- la media delle presenze escludendo i mesi estivi.

9 Si vogliono riordinare gli elementi di un vettore di *N* numeri interi generati casualmente con range (–20, +20) in modo che tutti i valori negativi stiano nella parte sinistra del vettore e tutti i valori non negativi stiano nella parte destra del vettore, senza utilizzare un array di supporto.

10 Inserisci un array di numeri reali non negativi (un numero negativo indica la fine della fase di inserimento). L'elaboratore calcola la seguente formula:

$$v[i] := (v[i-1] + v[i] + v[i+1]) / 3$$

e visualizza l'array così filtrato.

11 Scrivi un programma che, acquisiti due vettori di *N* numeri interi da input, **`vet1`** e **`vet2`**, stampi in output la differenza **`vet1 – vet2`**, cioè gli elementi di vet1 che non sono in **`vet2`**.

12. Scrivi un programma che genera casualmente *N* numeri di valore minore di 100 (dove *N* è una costante manifesta), li memorizza in un vettore e ricerca il numero che è maggiormente ripetuto nel vettore.
Traccia per la soluzione: si consiglia di utilizzare un secondo vettore di 100 elementi usando ogni cella come contatore delle ripetizioni dei numeri presenti nel primo vettore. Quindi si cerca l'elemento che alla fine contiene il valore più alto: il suo indice è il numero richiesto.

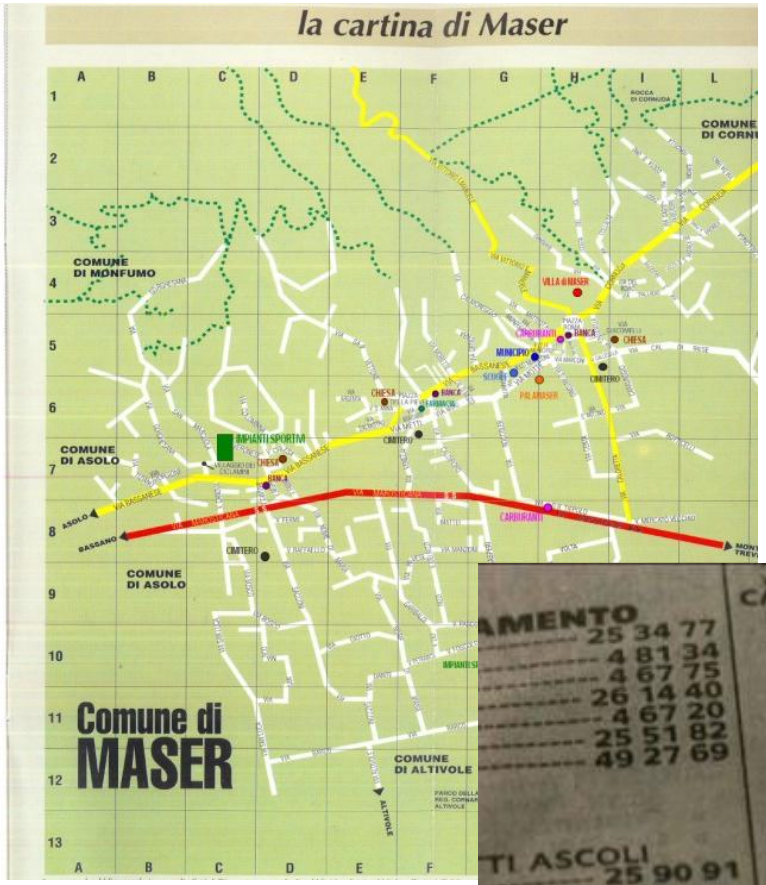
13 Leggi un insieme di numeri interi e inseriscilo in un array. L'insieme sarà chiuso dal numero zero. Successivamente, trova il numero massimo, il numero minimo e la loro posizione con meno di 2 $(n-1)$ confronti (precisamente con 3 /2 $[n-2]$ confronti).

Ordinamento di un vettore

Ordinamento di un vettore

- L'ordinamento di un vettore è strettamente correlato al problema di ricerca di un elemento nel vettore.
- È più facile trovare qualcosa in un insieme ordinato, infatti, per esempio:
  - È facile trovare una persona nell'elenco telefonico poiché è ordinato
  - È più facile trovare una via/piazza di una città consultando l'indice alfabetico delle vie (stradario) allegato alla piantina, piuttosto che cercarla sulla cartina stessa

Esempio: stradario ed elenco telefonico



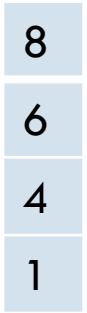
Stradario

Elenco telefonico

Criteri di ordinamento

- I criteri di ordinamento di un vettore possono essere due:
  - Ordinamento per valore crescente, cioè ogni elemento precede ("viene prima") ogni altro elemento che ha un valore più grande

  - Ordinamento per valore decrescente, cioè ogni elemento segue ("viene dopo") ogni altro elemento che ha un valore più grande

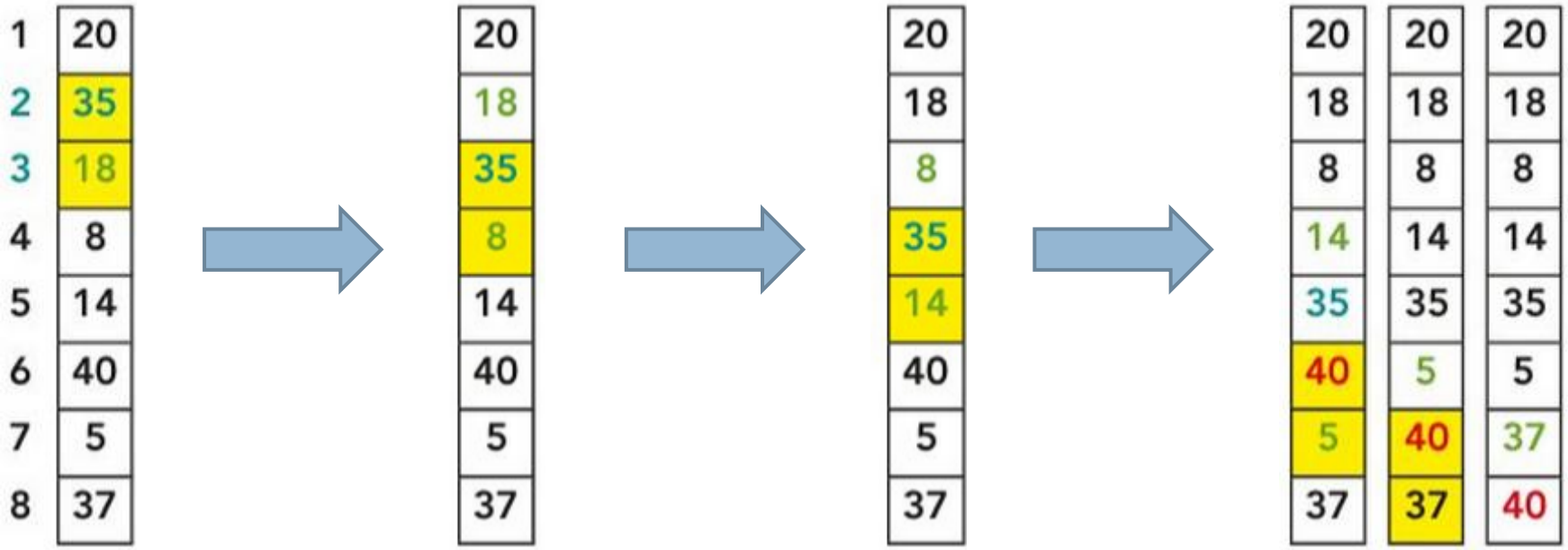


Algoritmi di ordinamento

- Gli algoritmi che ordinano un vettore sono molteplici, quello più famoso è il **Bubble-sort**, così chiamato perché assomiglia al movimento di bolle immerse in un liquido che si spostano verso l'alto o verso il basso a seconda del loro peso.
- Vedremo:
  - Bubble-sort
  - Bubble-sort con sentinella

Bubble-sort

- In caso di ordinamento in senso crescente l'algoritmo bubble-sort consiste nello scambiare di posto due valori se il primo è maggiore del secondo.



Bubble-sort

- Affiancando le colonne si nota che i numeri più "pesanti" (grandi) si spostano velocemente verso il basso mentre quelli più "leggeri" (piccoli) si spostano verso l'alto di una posizione.

1	20	20	20	20	20	20
2	35	18	18	18	18	18
3	18	35	8	8	8	8
4	8	8	35	14	14	14
5	14	14	14	35	35	35
6	40	40	40	40	5	5
7	5	5	5	5	40	37
8	37	37	37	37	37	40

Prima iterazione

Bubble-sort

- Facendo un altro passaggio (dalla prima posizione fino all'ultima posizione) si ha:

1	20	18	18	18	18
2	18	20	8	8	8
3	8	8	20	14	14
4	14	14	14	20	20
5	35	35	35	35	5
6	5	5	5	5	35
7	37	37	37	37	37
8	40	40	40	40	40

Seconda iterazione

Bubble-sort

- Facendo un'altra iterazione si ha:

Terza iterazione

Bubble-sort

- Facendo ulteriori iterazioni si ottiene il vettore ordinato.

Bubble-sort

- Possiamo fare due considerazioni:
  - Ad ogni iterazione poiché i numeri “leggeri” si spostano verso l’alto di una posizione, se il numero più piccolo si trova in ultima posizione in un vettore di N posizioni saranno necessarie N iterazioni per portarlo in prima posizione
  - A ogni iterazione i numeri “pesanti” scendono verso il basso fino a che non trovano un altro numero più “pesante” o fino a che non raggiungono il fondo

Algoritmo Bubble-sort

```
0  program bubble-sort;
1  (* ordinamento a bolle di un vettore *)
2  var
3    conta,coppie,tempo:integer;
4    vettore:array[1..20]of integer;
5  begin
6  (* riempimento casuale del vettore *)
7    for conta:=1 to 20 do
8      vettore[conta]:=random(100);
9  (* per i 20 elementi del vettore *)
```

Algoritmo Bubble-sort (continuo)

```
10   for conta:=1 to 20 do                          // ciclo esterno
11     (* per tutte le coppie *)
12     for coppie:=1 to 19 do                                  // ciclo interno
13       if vettore[coppie]>vettore[coppie+1]                  // confrontali
14         then
15           begin
16             tempo:=vettore[coppie];                         // scambiali
17             vettore[coppie]:=vettore[coppie+1]
18             vettore[coppie+1]:=tempo;
19           end;
20 (* visualizzazione del risultato *)
21   writeln('il vettore ordinato e'' il seguente');
22   for conta:=1 to 20 do
23     writeln(vettore[conta]);
24 end.
```

Bubble-sort con sentinella

- Poiché ad ogni iterazione l'elemento più "pesante" viene sistemato non c'è bisogno di andare a ordinare tutto il vettore, ma solamente la parte restante non ordinata
- Il Bubble-sort esegue tutte le istruzioni senza verificare se il vettore è già ordinato. Infatti se un vettore è già ordinato il Bubble-sort esegue tutti i controlli come se fosse disordinato
- Il Bubble-sort con sentinella "tiene memoria" di eventuali scambi effettuati. Se non ci sono stati scambi allora il vettore è ordinato

Algoritmo Bubble-sort con sentinella

```
program boubble_sort_con_sentinella;
const
     dim=5;
var
   vett : array[1..dim]of integer;
   i,temp : integer;
   scambio : boolean;

begin
     scambio := true;

     writeln('inserisci ', dim,' valori ');
     for i := 1 to dim do
         readln(vett[i]);
```

Dichiarazione delle variabili

Inizializzazione e assegnazione valore alle variabili

Algoritmo Bubble-sort con sentinella

```
repeat
        scambio := false;
        for i := 1 to dim-1 do
        begin
           if vett[i] > vett[i+1] then
           begin
                temp := vett[i];
                vett[i] := vett[i+1];
                vett[i+1] := temp;
                scambio := true;
           end;
        end;
until (scambio = false);
```

Scandisce tutti i valori del vettore

Se l'elemento corrente è maggiore del successivo li scambia di posto

Il ciclo si ripete solo se viene scambiato almeno un valore

Algoritmo Bubble-sort con sentinella

```
    writeln('I valori ordinati sono: ');
    for i := 1 to dim do
        writeln(vett[i]);
    readln;
end.
```

Stampa a monitor del vettore ordinato

- **Esercizio**: riscrivi l'algoritmo utilizzando il costrutto **while..do**

Riepilogo

- **Ordinare** significa disporre un insieme di oggetti omogenei secondo un criterio che stabilisce la modalità di disposizione di due elementi adiacenti dell'insieme stesso.
- È un problema fondamentale dell'informatica, assieme al problema della ricerca di un elemento.
- I criteri che regolano l'ordinamento degli elementi sono due:
  ordinamento per valore **crescente** (senso crescente);
  ordinamento per valore **decrescente** (senso decrescente).
- Un semplice algoritmo di ordinamento l'algoritmo per scambio, o **bubble sort**, che continua a **effettuare** il controllo di due celle adiacenti eventualmente scambiandone il contenuto i base al loro valore.
  L'algoritmo termina quando in un ciclo completo non sono stati fatti scambi: a tal fine ci si avvale di una variabile booleana (sentinella) da utilizzare come condizione di terminazione del ciclo.

Esercizi 1-6, pag. 403

1 Scrivi un programma che genera casualmente 30 numeri di valore minore di 1000, li memorizza in un vettore e visualizza dal più piccolo al più grande i numeri pari.

2 Scrivi un programma che genera casualmente *N* numeri di valore minore di 100 (dove *N* è una costante manifesta), li memorizza in un vettore e cancella i numeri doppi presenti sostituendoli con 0. Quindi li ordina in senso decrescente.

3 Scrivi un programma che genera casualmente *N* numeri di valore minore di 100 (dove *N* è una costante manifesta), li memorizza in un vettore e cancella i numeri doppi presenti sostituendoli con 0. Quindi li ordina in senso crescente ignorando il numero 0 e li visualizza sullo schermo.

4 Scrivi un programma che genera casualmente *N* numeri di valore minore di 1000 (dove *N* è una costante manifesta), li memorizza in un vettore e cancella i numeri inferiori di un numero MAX letto da tastiera. Quindi visualizza il vettore ordinato in senso crescente.

5 Riordina gli elementi di un vettore di *N* numeri interi generati casualmente con range (–20,+20) in modo che tutti i valori negativi siano copiati in un nuovo vettore NEGATIVI e tutti i valori non negativi vengano messi in un vettore POSITIVI. Quindi visualizza i due gruppi di dati ordinati in senso crescente.

6 Riordina gli elementi di un vettore di 30 numeri interi generati casualmente con range (–30,+30) in modo che tutti i valori negativi siano copiati in un nuovo vettore NEGATIVI e tutti i valori non negativi vengano messi in un vettore POSITIVI escludendo i numeri doppi e generando al posto di questi un nuovo numero in sostituzione. Quindi visualizza i due gruppi di dati ordinati in senso decrescente.

7 In un array di 10 elementi sono memorizzati, ordinati, i 10 migliori tempi ottenuti dagli atleti in una gara di discesa libera. Si riceve poi come dato d'ingresso il tempo ottenuto da un nuovo concorrente: verifica la sua appartenenza fra i 10 migliori risultati e, in caso affermativo, inseriscilo nella corretta posizione all'interno del vettore.
Traccia per la soluzione: prima si individua la posizione in cui deve essere inserito il dato scorrendo il vettore da sinistra verso destra, quindi si crea un "buco" dove inserirlo, cioè si libera una cella, spostando a destra di una posizione tutti i numeri da quella posizione fino ad arrivare all'estremo del vettore; naturalmente si perde il valore dell'ultima cella, che esce dalla classifica.

Esercizi 7-11, pag. 403

8 Effettua l'ordinamento di un vettore mediante un algoritmo "per enumerazione": conoscendo il valore massimo degli elementi presenti in un vettore, esegui un ciclo a conteggio con tale valore come estremo superiore e, a ogni iterazione, verifica se il valore corrente dell'indice è presente nel vettore da ordinare; in tal caso copialo in un nuovo vettore.

9 Effettua l'ordinamento di un vettore (con elementi tutti diversi) mediante un algoritmo counting-sort: conoscendo il valore massimo degli elementi presenti in un vettore, per ogni elemento calcola quanti altri elementi presenti nel vettore hanno valore minore; posiziona quindi l'elemento corrente in un nuovo vettore dopo aver lasciato tanti spazi vuoti quanti sono gli elementi calcolati in precedenza.

10 Scrivi un programma che costruisce un vettore di interi a partire dalle informazioni contenute in altri due vettori: il nuovo vettore viene costruito in modo tale per cui gli elementi in esso contenuti siano in ordine crescente partendo da due vettori anch'essi ordinati e deve essere realizzato per merge (o fusione).

11 Realizza un programma che riempia casualmente un vettore per n volte per ciascuna delle seguenti dimensioni:

- 20 elementi;
- 50 elementi;
- 100 elementi;
- 500 elementi;
- 1000 elementi.

Utilizza tutti gli algoritmi di ordinamento conosciuti e crea una tabella che permetta di effettuare la comparazione dei tempi di elaborazione di tali algoritmi, calcolando la media dei tempi di elaborazione sulle n volte che viene generato il vettore della medesima dimensione.
Produci infine una tabella con la media generale dei tempi in modo da ottenere un unico indice che classifichi gli algoritmi indipendentemente dalla dimensione del problema.