Programmazione in C

Unità
Vettori

Vettori

- Strutture dati complesse
- I vettori in C
- Operazioni elementari sui vettori
- Esercizi guidati sui vettori
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitoli 1 e 5
  - Cabodi, Quer, Sonza Reorda: capitolo 5
  - Dietel & Dietel: capitolo 6
- Dispense
  - Scheda: “Vettori in C”

Vettori

Strutture dati complesse

Strutture dati complesse

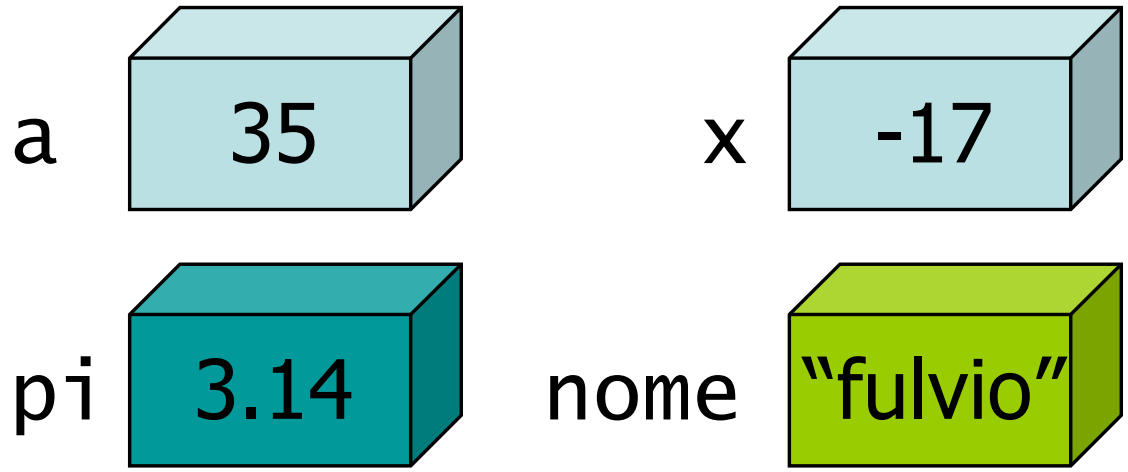
- Tipi di dato strutturati
- Introduzione ai vettori
- Caratteristiche dei vettori

Strutture dati complesse

Tipi di dato strutturati

Tipi di dato strutturati

- Finora abbiamo utilizzato dei tipi di dato semplici
  - `int`, `float`
  - Ogni variabile può contenere **un solo** valore
- Il linguaggio C permette di definire tipi di dato complessi, aggregando più variabili semplici

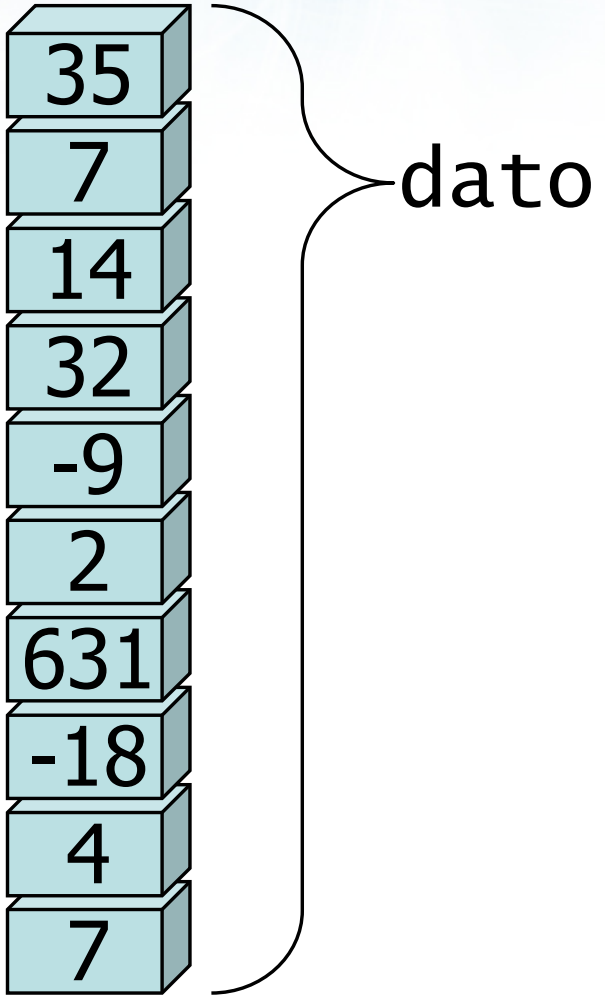


Esigenze

- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro

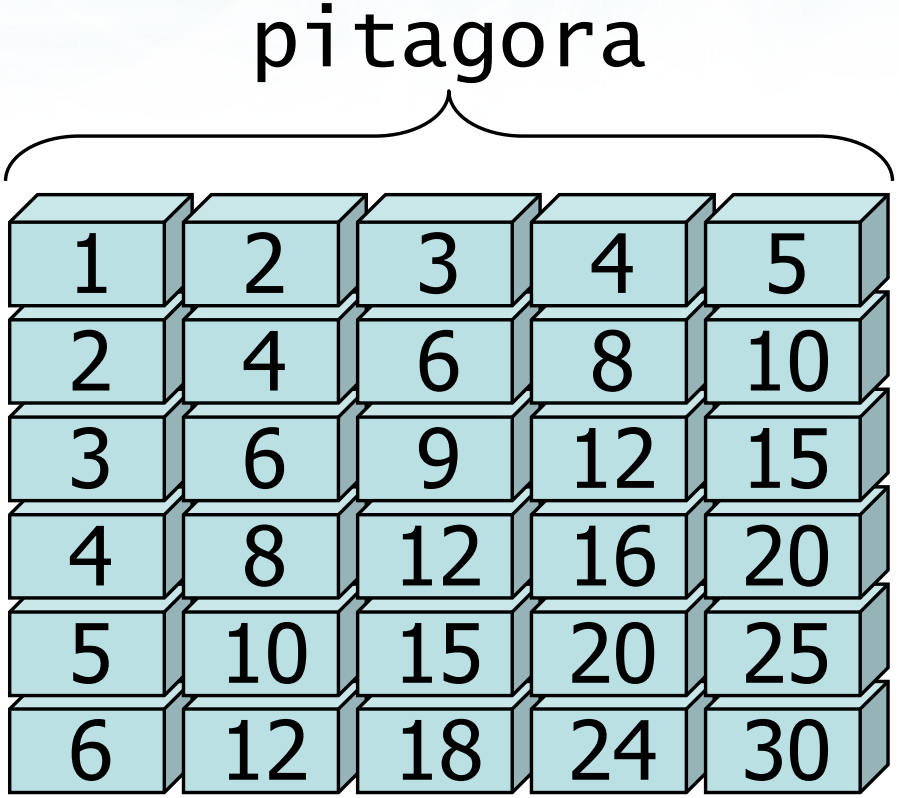
Esigenze

- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro



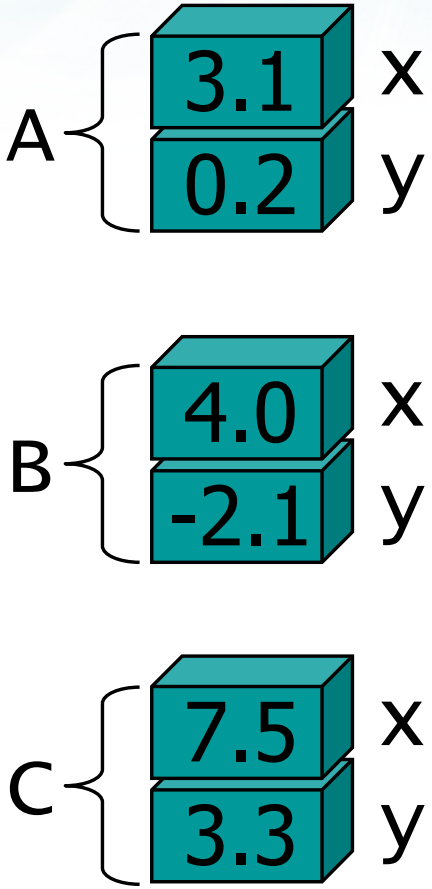
Esigenze

- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro



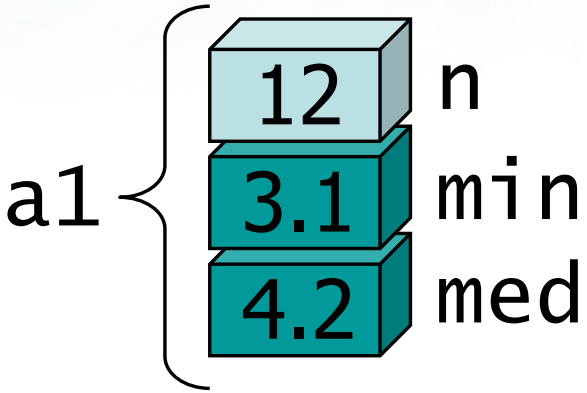
Esigenze

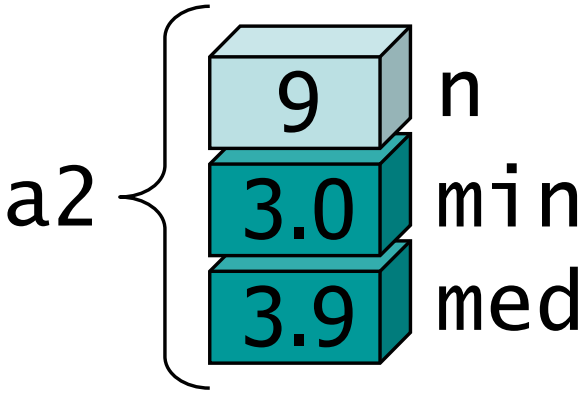
- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro



Esigenze

- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro





Dati strutturati (1/2)

- Raggruppare più variabili semplici in un'unica struttura complessa
- Diversi meccanismi di aggregazione
  - Vettore
  - Matrice
  - Struttura

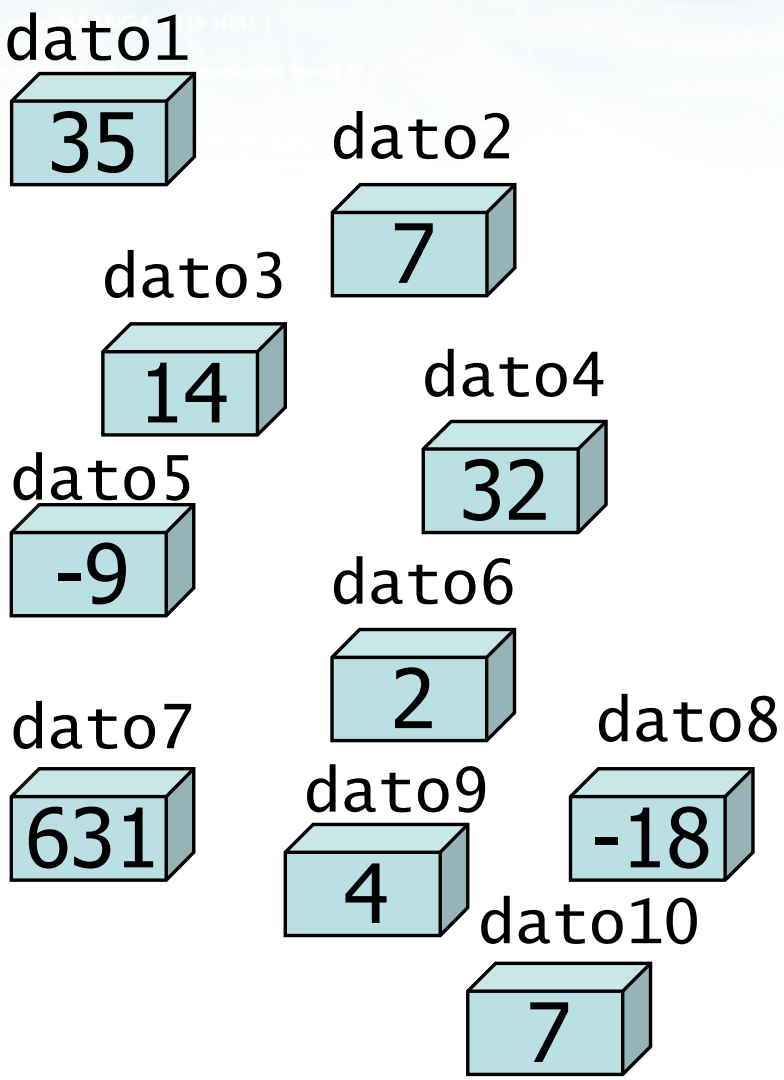
Dati strutturati (2/2)

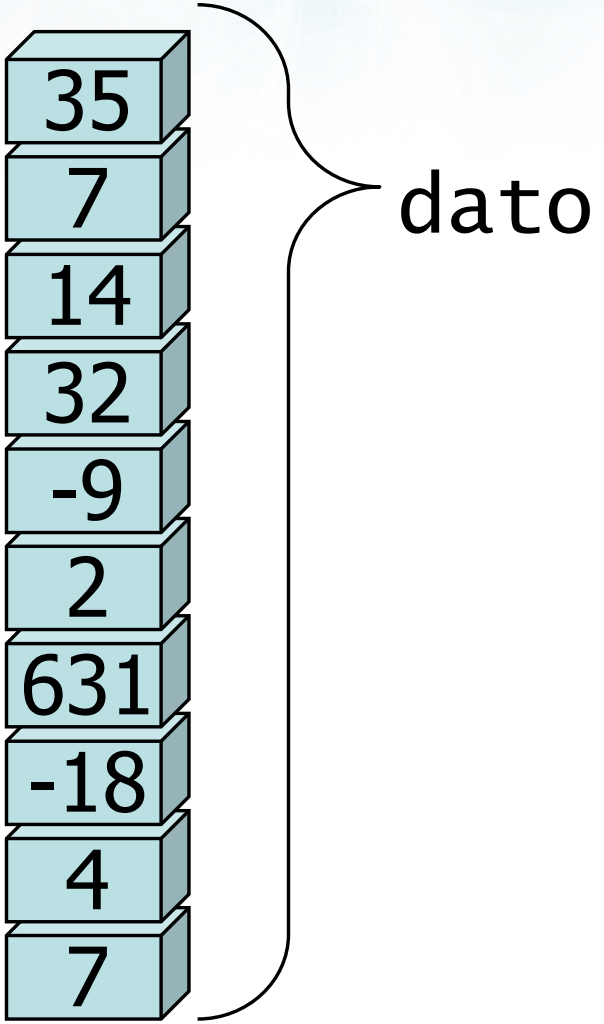
- Una sola variabile, di tipo complesso, memorizza **tutti** i dati della struttura complessa
  - `Vettore_di_int dato`
  - `Matrice_di_int pitagora`
  - `Struttura_xy A, B, C`
  - `Struttura_auto a1, a2`

Strutture dati complesse

Introduzione ai vettori

Variabili e vettori





Da evitare...

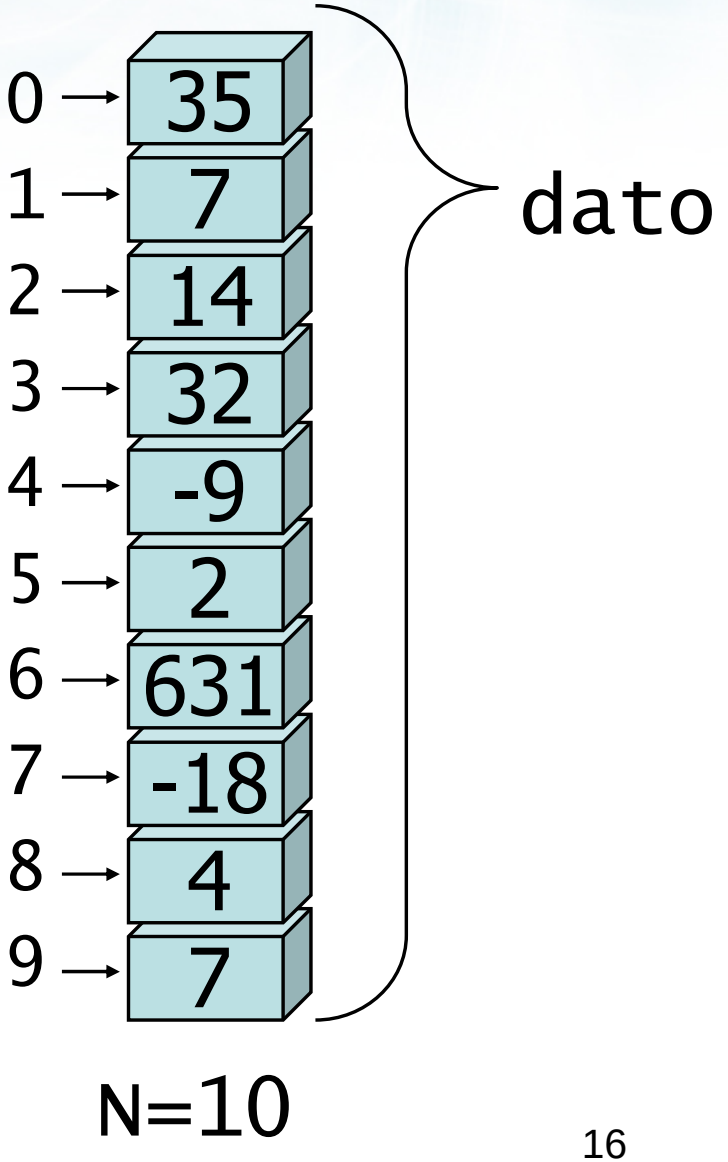
```
int main(void)
{
    int dato1, dato2, dato3, dato4, dato5 ;
    int dato6, dato7, dato8, dato9, dato10 ;
    . . . . .
    scanf("%d", &dato1) ;
    scanf("%d", &dato2) ;
    scanf("%d", &dato3) ;
    . . .
    scanf("%d", &dato10) ;

    printf("%d\n", dato10) ;
    printf("%d\n", dato9) ;
    printf("%d\n", dato8) ;
    . . .
    printf("%d\n", dato1) ;
}
```

Vettori

- Meccanismo di strutturazione più semplice
- Utilizzato per aggregare serie di dati
- Permette di memorizzare tutti i dati acquisiti o calcolati, e potervi accedere
  - In qualsiasi momento
  - In qualsiasi ordine
- I singoli dati sono distinti dal loro numero d'ordine
  - Primo, secondo, ..., ultimo dato
  - Ciascun dato può avere un valore diverso

Vettori

- Sequenza lineare di dati elementari
- Elementi tutti dello stesso tipo
- Numero di elementi fisso (N)
- Elementi identificati dal proprio indice
  - da 0 a N-1



...così è meglio!

```
int main(void)
{
    int dato[10] ;
    . . . . .
    for( i=0; i<10; i++)
       scanf("%d", &dato[i]) ;

    for( i=9; i>=0; i--)
       printf("%d\n", dato[i]) ;
}
```

Insieme o separati?

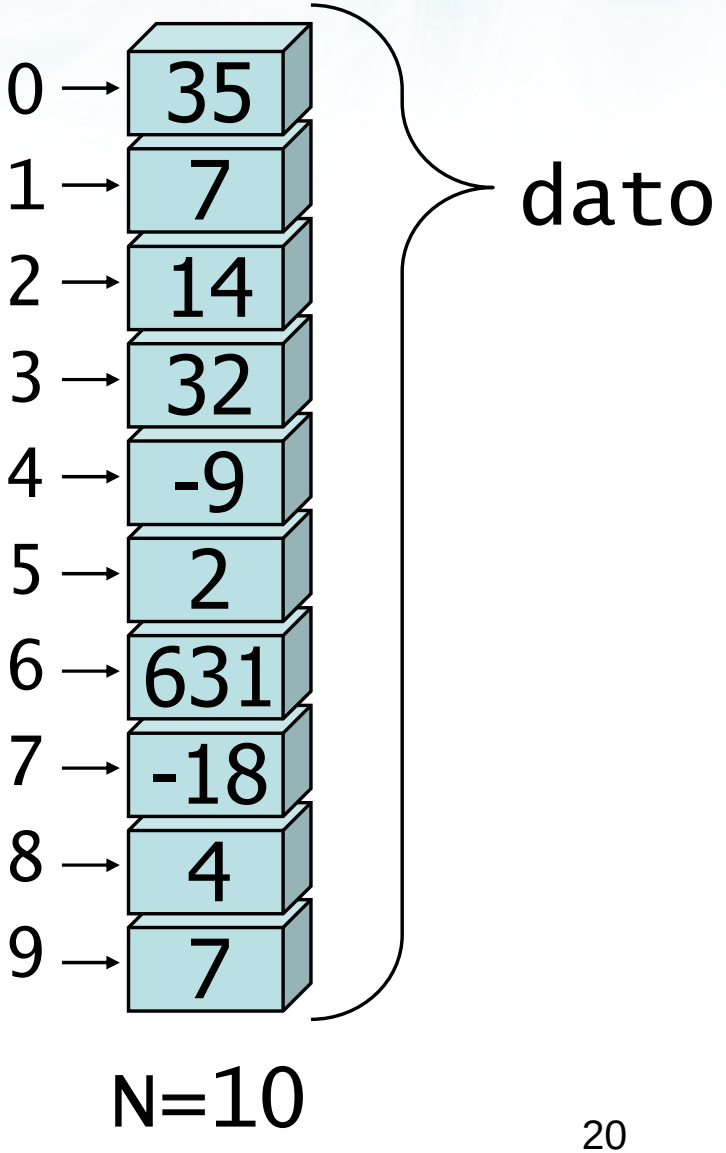
- I singoli dati vengono tenuti insieme in un'**unica variabile** di tipo vettore
  - `int dato[10]`
- Le operazioni (lettura, scrittura, elaborazione) avvengono però sempre **un dato alla volta**
  - `scanf("%d", &dato[i])`
  - `printf("%d\n", dato[i])`
- Ogni singolo dato è identificato da
  - Nome del vettore
  - Posizione all'interno del vettore

Strutture dati complesse

Caratteristiche dei vettori

Caratteristiche dei vettori

- Caratteristiche statiche
  - Nome
    - dato
  - Tipo di dato base
    - int
  - Dimensione totale
    - 10
- Caratteristiche dinamiche
  - Valori assunti dalle singole celle
    - 35, 7, 14, 32, ...



Esempio

- Leggere da tastiera 10 numeri e stamparli in ordine inverso
  - Tipo base: int
  - Dimensione: 10
  - Lettura delle celle da 0 a 9
  - Stampa delle celle da 9 a 0

Vincoli

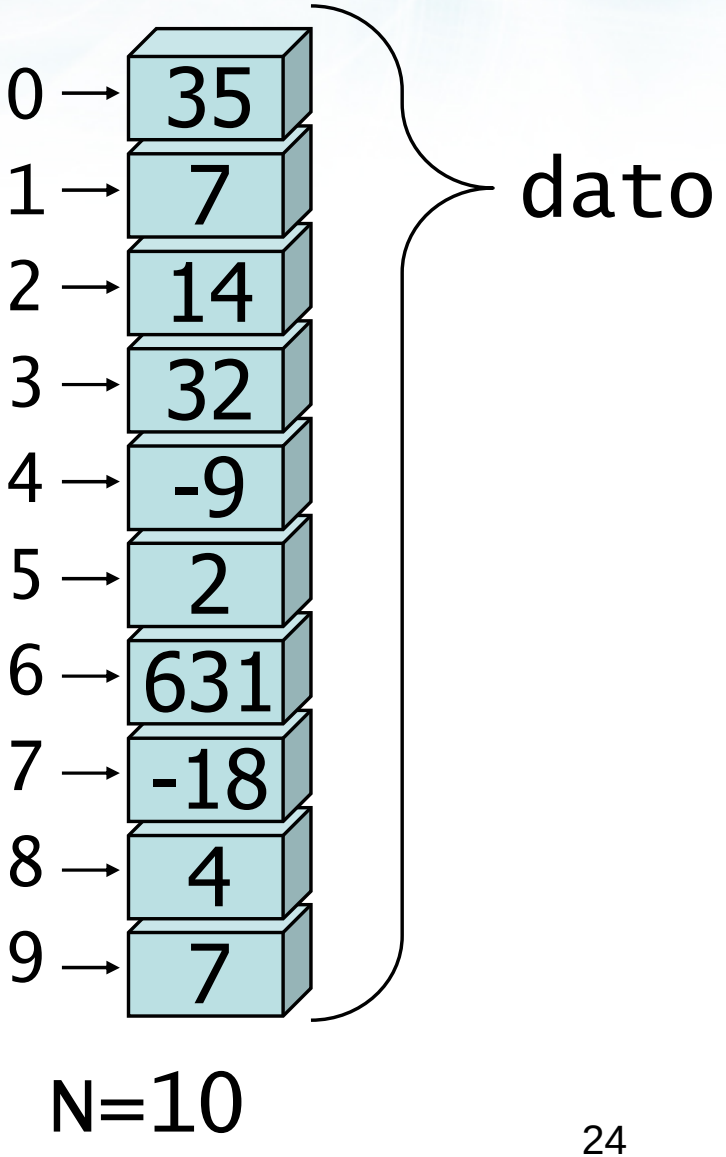
- Tutte le celle di un vettore avranno lo **stesso nome**
- Tutte le celle di un vettore devono avere lo **stesso tipo base**
- La dimensione del vettore è **fissa** e deve essere determinata al momento della sua definizione
  - La dimensione è sempre un numero intero
- Ogni cella ha **sempre** un valore
  - Impossibile avere celle "vuote"
  - Le celle non inizializzate contengono valori ignoti

Accesso alle celle

- Ciascuna cella è identificata dal proprio **indice**
- Gli indici sono sempre numeri **interi**
  - In C, gli indici partono da **0**
- Ogni cella è a tutti gli effetti una **variabile** il cui tipo è pari al tipo base del vettore
- Ogni cella, indipendentemente dalle altre
  - deve essere inizializzata
  - può essere letta/stampata
  - può essere aggiornata da istruzioni di assegnazione
  - può essere usata in espressioni aritmetiche

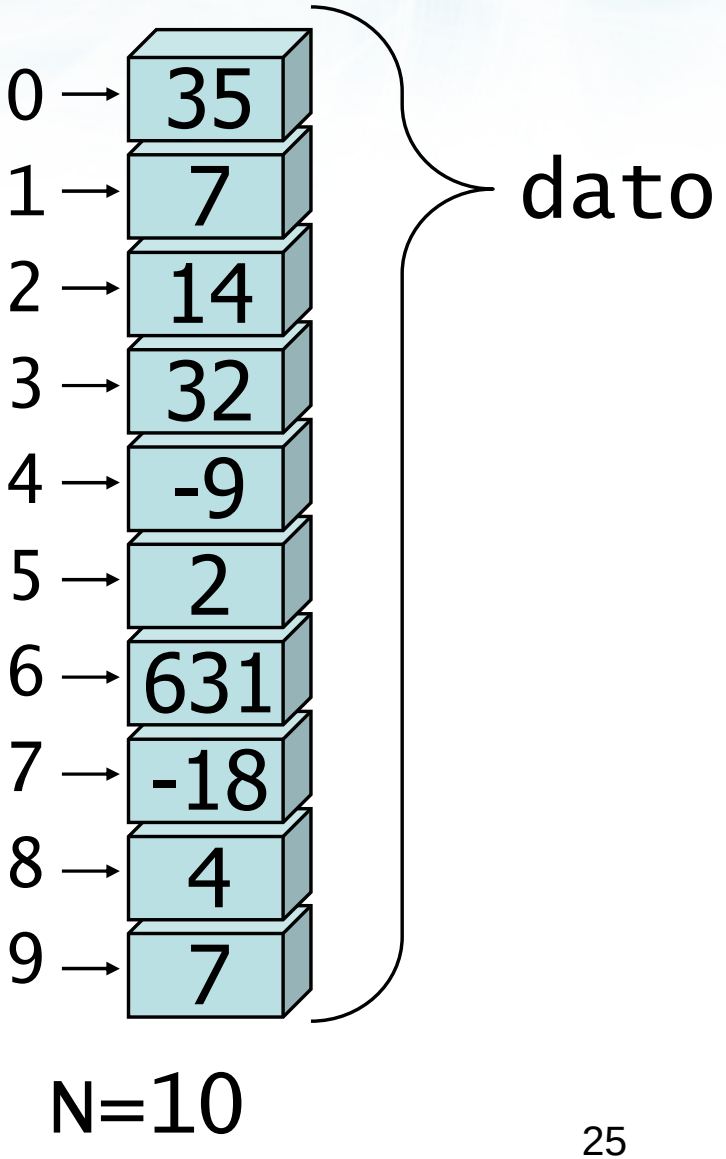
Errore frequente

- Non confondere mai l'indice con il contenuto
  - `dato[5] = 2`
  - `dato[9] == dato[1]`
  - `dato[i]>dato[j]`
  - `i>j`



Errore frequente

- **Non si può** effettuare alcuna operazione sul vettore
  - `dato = 0`
  - `printf("%d", dato)`
- Occorre operare sui singoli elementi
  - Solitamente all'interno di un ciclo `for`



Vettori

I vettori in C

I vettori in C

- Sintassi della definizione
- Definizione di costanti
- Operazioni di accesso

I vettori in C

Sintassi della definizione

Definizione di vettori in C

```
int dato[10] ;
```

Tipo di dato base

Nome del vettore

Numero di elementi

Definizione di vettori in C

```
int dato[10] ;
```

Tipo di dato base

Nome del vettore

Numero di elementi

- Stesse regole che valgono per i nomi delle variabili.
- I nomi dei vettori devono essere diversi dai nomi delle variabili.

Definizione di vettori in C

```
int dato[10] ;
```

Tipo di dato base

Nome del vettore

Numero di elementi

- `int`
- `float`
- In futuro vedremo: `char`, `struct`

Definizione di vettori in C

```
int dato[10] ;
```

Tipo di dato base

Nome del vettore

Numero di elementi

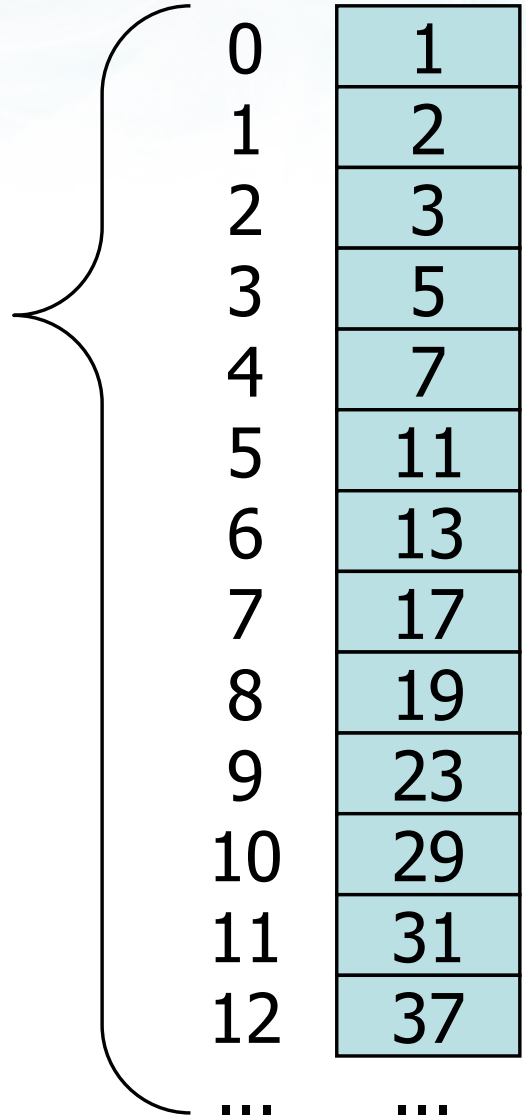
- Intero positivo
- Costante nota a tempo di compilazione
- Impossibile cambiarla in seguito

Esempi

- `int dato[10] ;`
- `float lati[8] ;`
- `int numeriprimi[100] ;`
- `int is_primo[1000] ;`

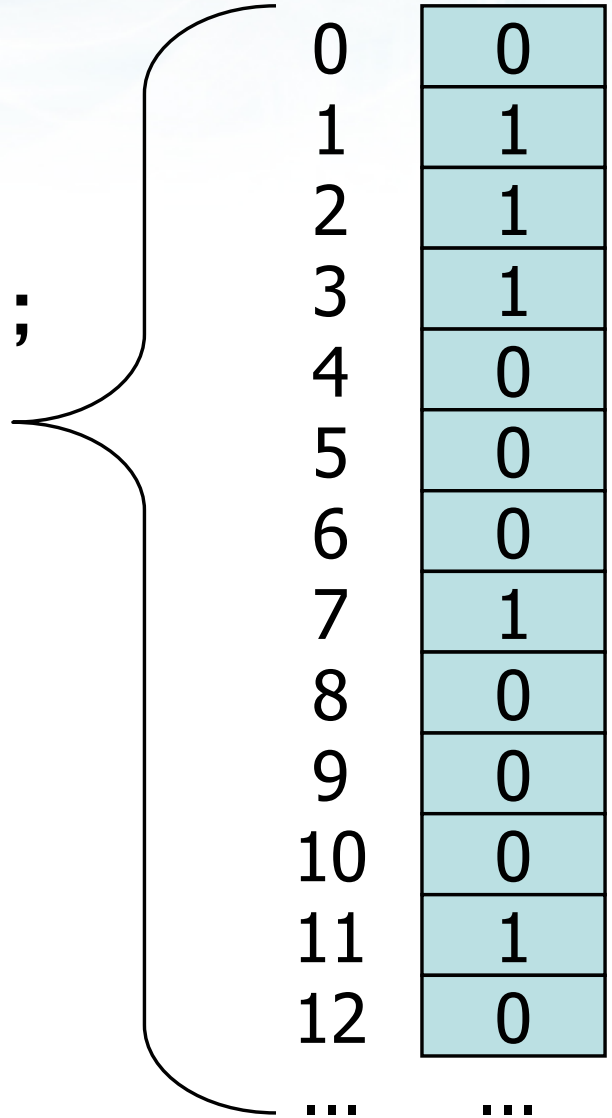
Esempi

- `int dato[10] ;`
- `float lati[8] ;`
- `int numeriprimi[100] ;`
- `int is_primo[1000] ;`



Esempi

- `int dato[10] ;`
- `float lati[8] ;`
- `int numeriprimi[100] ;`
- `int is_primo[1000] ;`



Errore frequente

- Dichiarare un vettore usando una variabile anziché una costante

```
int N = 10 ;
int dato[N] ;
```

```
int dato[10] ;
```

Errore frequente

- Dichiarare un vettore usando una variabile non ancora inizializzata

```
int N ;
int dato[N] ;

. . .

scanf("%d",&N) ;
```

```
int dato[10] ;
```

Errore frequente

- Dichiarare un vettore usando il nome dell'indice

```
int i ;
int dato[i] ;

. . .
for(i=0; i<10; i++)
  scanf("%d",&dato[i]) ;
```

```
int dato[10] ;
```

I vettori in C

Definizione di costanti

Costanti

- La dimensione di un vettore deve essere specificata utilizzando una costante intera positiva
  - **Costante** = valore numerico già noto al momento della compilazione del programma

```
int dato[10] ;
```

Problemi

```
int i ;
int dato[10] ;

. . .

for(i=0; i<10; i++)
  scanf("%d", &dato[i]) ;

for(i=0; i<10; i++)
  printf("%d\n", dato[i]) ;
```

Devono essere tutte uguali. Chi lo garantisce?

Problemi

```
int i ;
int dato[10] ;

. . .

for(i=0; i<10; i++)
  scanf("%d", &dato[i]) ;

for(i=0; i<10; i++)
  printf("%d\n", dato[i]) ;
```

Se volessi lavorare con 20 dati dovrei modificare in tutti questi punti.

Soluzione

- Per risolvere i problemi visti si può ricorrere alle **costanti simboliche**
  - Associamo un nome simbolico ad una costante
  - Nel programma usiamo sempre il nome simbolico
  - Il compilatore si occuperà di sostituire, ad ogni occorrenza del nome, il valore numerico della costante

Costanti simboliche

- Costrutto `#define`
  - Metodo originario, in tutte le versioni del C
  - Usa una sintassi particolare, diversa da quella del C
  - Definisce costanti valide su tutto il file
  - Non specifica il tipo della costante
- Modificatore `const`
  - Metodo più moderno, nelle versioni recenti del C
  - Usa la stessa sintassi di definizione delle variabili
  - Specifica il tipo della costante

Costrutto #define

```
#define N 10

int main(void)
{
    int dato[N] ;
    . . .
}
```

Definizione della costante

Uso della costante

Particolarità (1/2)

- La definizione non è terminata dal segno ;
- Tra il nome della costante ed il suo valore vi è solo uno spazio (non il segno =)
- Le istruzioni `#define` devono essere una per riga
- Solitamente le `#define` vengono poste subito dopo le `#include`

```
#define N 10
```

Particolarità (2/2)

- Non è possibile avere una `#define` ed una variabile con lo stesso nome
- Per convenzione, le costanti sono indicate da nomi TUTTI_MAIUSCOLI

```
#define N 10
```

Esempio

```
#define MAX 10

int main(void)
{
   int i ;
   int dato[MAX] ;

   . . .

   for(i=0; i<MAX; i++)
     scanf("%d", &dato[i]) ;

   for(i=0; i<MAX; i++)
     printf("%d\n", dato[i]) ;
}
```

Modificatore const

```
int main(void)
{
    const int N = 10 ;

    int dato[N] ;

    . . .
}
```

Definizione della costante

Uso della costante

Sintassi

- Stessa sintassi per dichiarare una variabile
- Parola chiave `const`
- Valore della costante specificato dal segno =
- Definizione terminata da segno `;`
- Necessario specificare il tipo (es. `int`)
- Il valore di `N` non si può più cambiare

```
const int N = 10 ;
```

Esempio

```
int main(void)
{
   const int MAX = 10 ;
   int i ;
   int dato[MAX] ;

   . . .

   for(i=0; i<MAX; i++)
      scanf("%d", &dato[i]) ;

   for(i=0; i<MAX; i++)
      printf("%d\n", dato[i]) ;
}
```

Suggerimenti

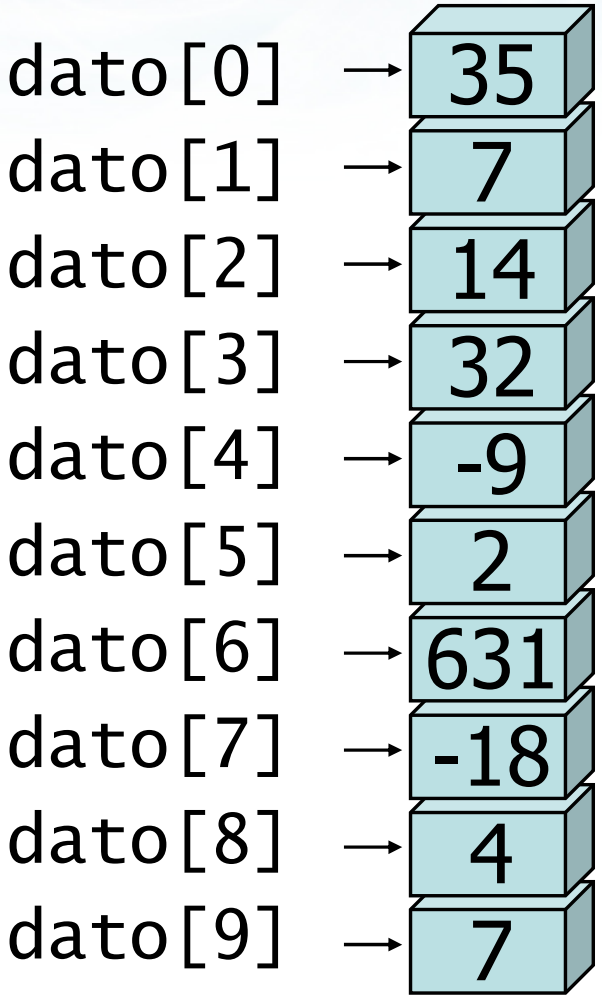
- Utilizzare sempre il modificatore `const`
- Permette maggiori controlli da parte del compilatore
- Gli eventuali messaggi d'errore sono più chiari
- Aggiungere sempre un commento per indicare lo scopo della variabile
- Utilizzare la convenzione di assegnare nomi TUTTI_MAIUSCOLI alle costanti

I vettori in C

Operazioni di accesso

Accesso ai valori di un vettore

- Ciascun elemento di un vettore è equivalente ad una singola variabile avente lo stesso tipo base del vettore
- È possibile accedere al contenuto di tale variabile utilizzando l'operatore di **indicizzazione**: `[ ]`




```
int dato[10] ;
```

Sintassi

```
nomevettore[ valoreindice ]
```

Come nella dichiarazione

Costante, variabile o espressione aritmetica con valore intero

Valore intero compreso tra 0 e (dimensione del vettore – 1)

Vincoli

- Il valore dell'indice deve essere compreso tra 0 e N-1. La responsabilità è del programmatore
- Se l'indice non è un numero intero, viene automaticamente troncato
- Il nome di un vettore può essere utilizzato solamente con l'operatore di indicizzazione

Uso di una cella di un vettore

- L'elemento di un vettore è utilizzabile come una qualsiasi variabile:
  - utilizzabile all'interno di un'espressione
    - `tot = tot + dato[i] ;`
  - utilizzabile in istruzioni di assegnazione
    - `dato[0] = 0 ;`
  - utilizzabile per stampare il valore
    - `printf("%d\n", dato[k]) ;`
  - utilizzabile per leggere un valore
    - `scanf("%d\n", &dato[k]) ;`

Esempi

- `if (dato[i]==0)`
  - se l'elemento contiene zero
- `if (dato[i]==dato[i+1])`
  - due elementi consecutivi uguali
- `dato[i] = dato[i] + 1 ;`
  - incrementa l'elemento `i`-esimo
- `dato[i] = dato[i+1] ;`
  - copia un dato dalla cella successiva

Vettori

Operazioni elementari sui vettori

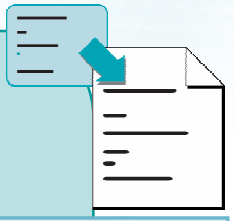
Operazioni elementari sui vettori

- Definizioni
- Stampa di un vettore
- Lettura di un vettore
- Copia di un vettore
- Ricerca di un elemento
- Ricerca del massimo o minimo
- Vettori ad occupazione variabile

Operazioni elementari sui vettori

Definizioni

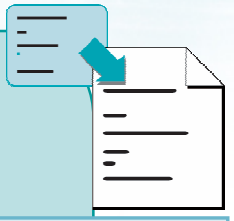
Definizioni (1/2)



vettori.c

```
const int N = 10 ;
   /* dimensioni dei vettori */

int v[N] ; /* vettore di N interi */

float r[N] ;
   /* vettore di N reali */

int i, j ;
   /* indici dei cicli */
```

Definizioni (2/2)



vettori.c

```
const int M = 100 ;
   /* dimensioni dei vettori */

int w[N] ; /* vettore di N interi */
int h[M] ; /* vettore di M interi */

int dato ;
   /* elemento da ricercare */
```

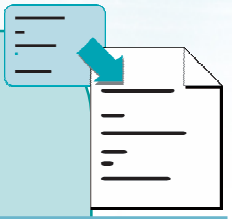
Operazioni elementari sui vettori

Stampa di un vettore

Stampa di un vettore

- Occorre stampare un elemento per volta, all'interno di un ciclo `for`
- Ricordare che
  - gli indici del vettore variano tra 0 e `N-1`
  - gli utenti solitamente contano tra 1 e `N`
  - `v[i]` è l'elemento `(i+1)`-esimo

Stampa vettore di interi

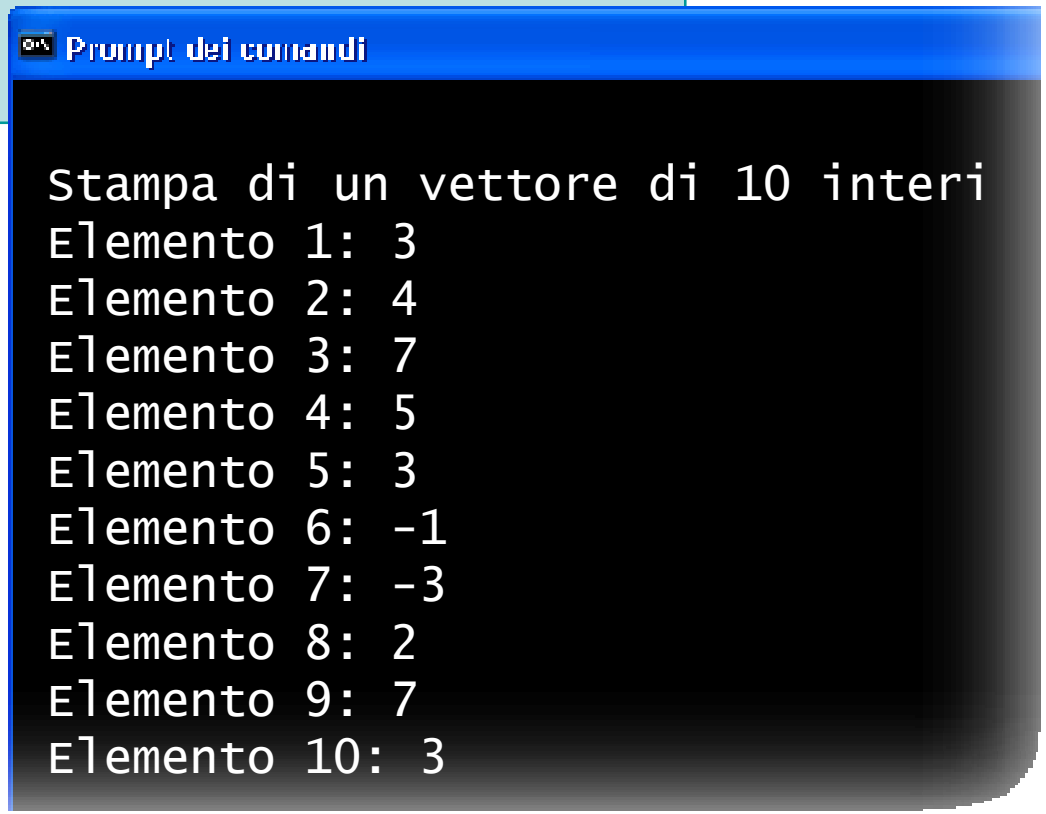


vettori.c

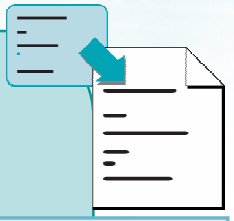
```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
     printf("Elemento %d: ", i+1) ;
     printf("%d\n", v[i]) ;
}
```

Stampa vettore di interi

```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    printf("%d\n", v[i]) ;
}
```



Stampa in linea

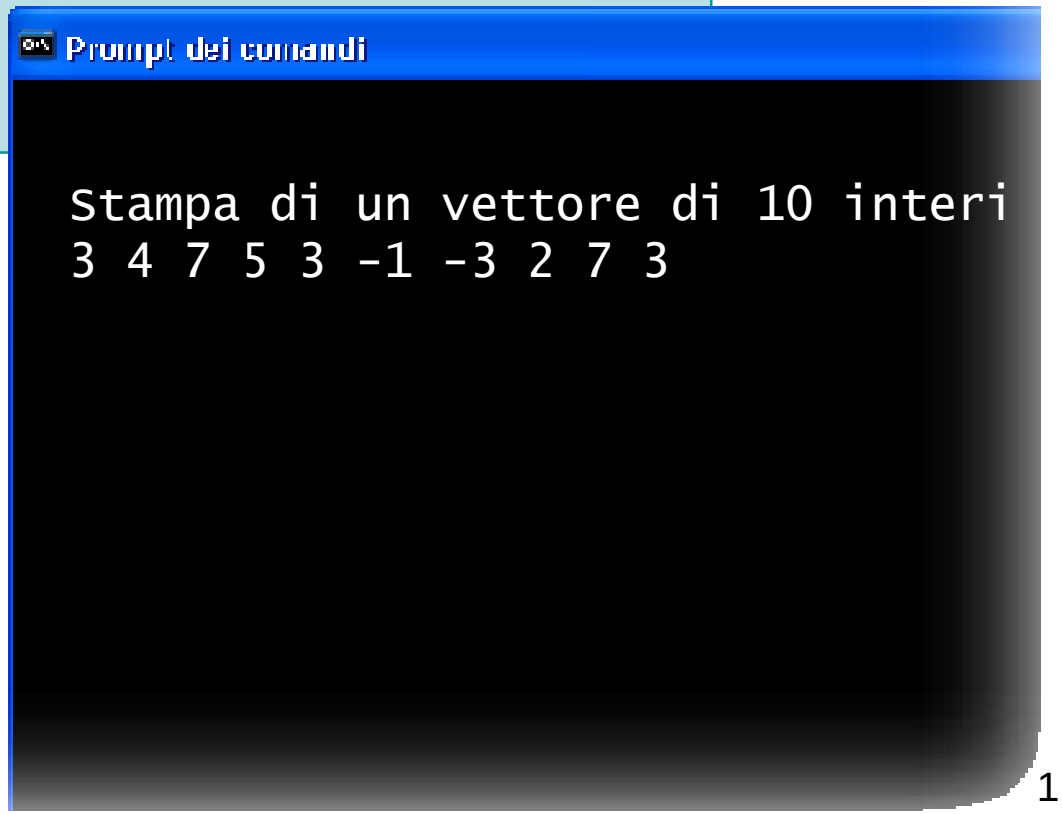


vettori.c

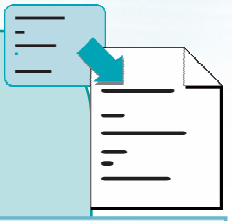
```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("%d ", v[i]) ;
}
printf("\n") ;
```

Stampa in linea

```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("%d ", v[i]) ;
}
printf("\n") ;
```



Stampa vettore di reali



vettori.c

```
printf("Vettore di %d reali\n", N) ;

for( i=0; i<N; i++ )
{
     printf("Elemento %d: ", i+1) ;
     printf("%f\n", r[i]) ;
}
```

Avvertenze

- Anche se il vettore è di reali, l'indice è sempre intero
- Separare sempre i vari elementi almeno con uno spazio (o un a capo)

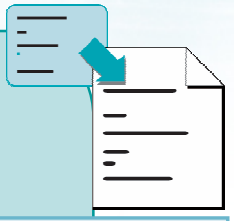
Operazioni elementari sui vettori

Lettura di un vettore

Lettura di un vettore

- Occorre leggere un elemento per volta, all'interno di un ciclo `for`
- Ricordare l'operatore & nella `scanf`

Lettura vettore di interi

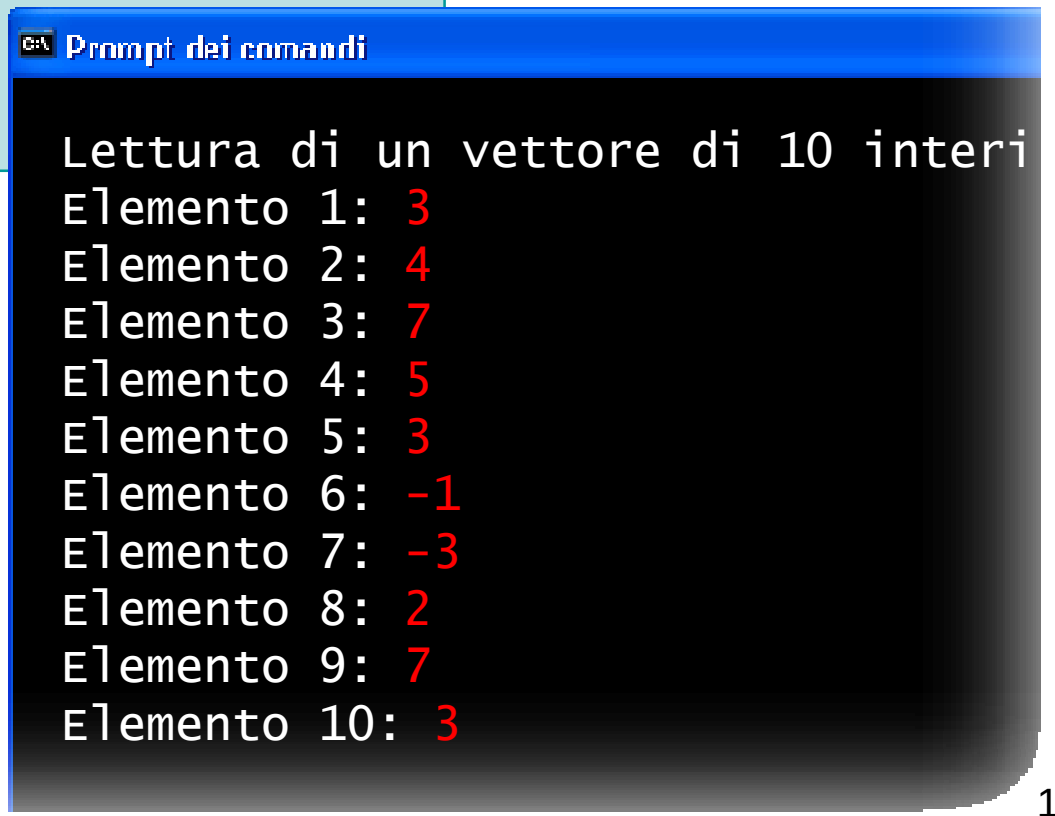


vettori.c

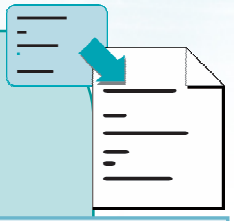
```
printf("Lettura di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    scanf("%d", &v[i]) ;
}
```

Lettura vettore di interi

```
printf("Lettura di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
     printf("Elemento %d: ", i+1) ;
     scanf("%d", &v[i]) ;
}
```



Lettura vettore di reali



vettori.c

```
printf("Lettura di %d reali\n", N) ;

for( i=0; i<N; i++ )
{
     printf("Elemento %d: ", i+1) ;
     scanf("%f", &r[i]) ;
}
```

Avvertenze

- Anche se il vettore è di reali, l'indice è sempre intero
- Fare precedere sempre ogni lettura da una `printf` esplicativa

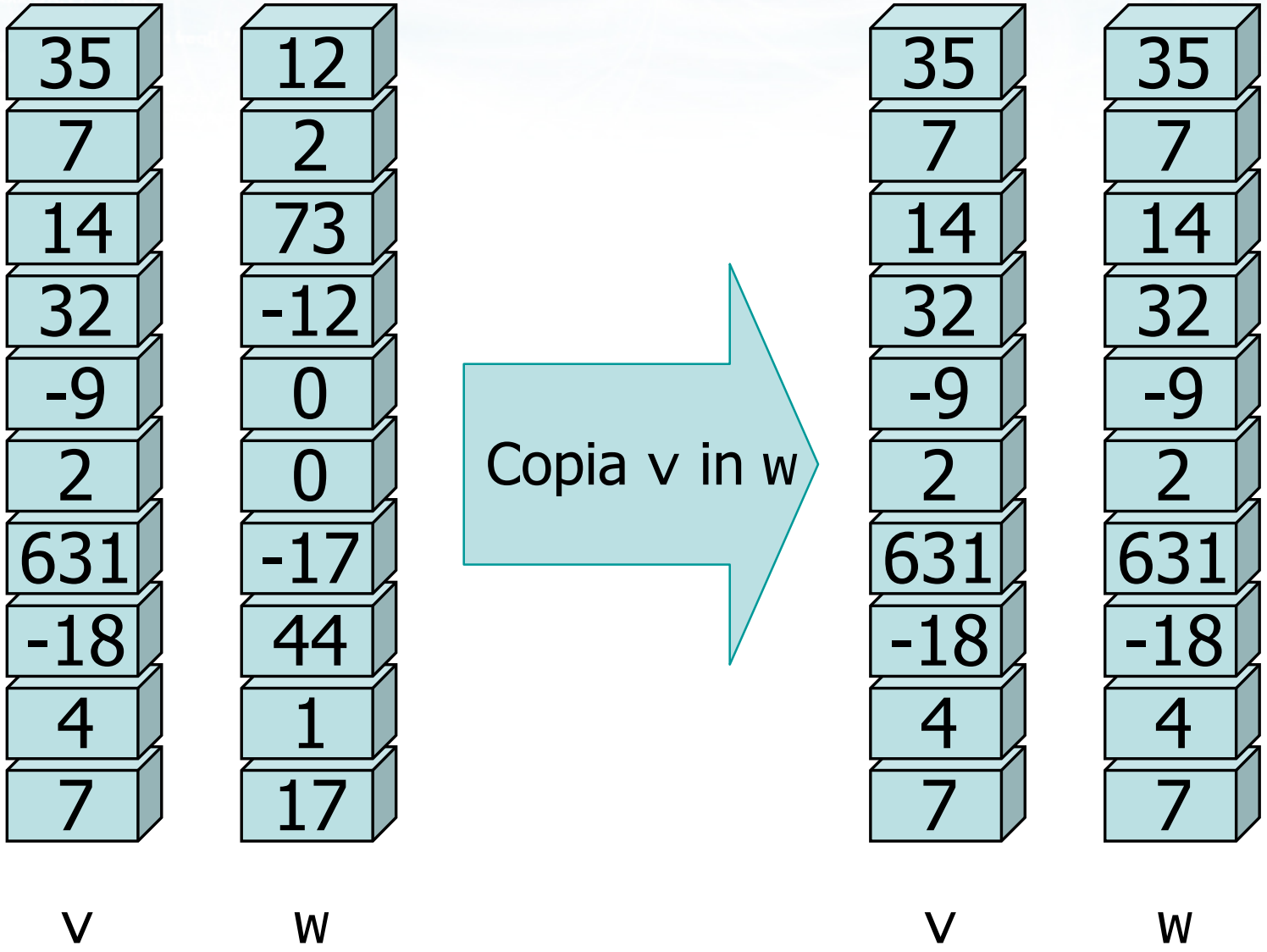
Operazioni elementari sui vettori

Copia di un vettore

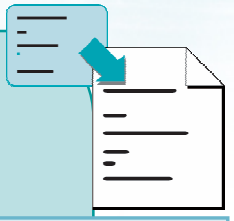
Copia di un vettore

- Più correttamente, si tratta si copiare **il contenuto** di un vettore in un altro vettore
- Occorre copiare un elemento per volta dal vettore "sorgente" al vettore "destinazione", all'interno di un ciclo `for`
- I due vettori devono avere lo stesso numero di elementi, ed essere dello stesso tipo base

Copia di un vettore



Copia di un vettore



vettori.c

```
/* copia il contenuto di v[] in w[] */

for( i=0; i<N; i++ )
{
    w[i] = v[i] ;
}
```

Avvertenze

- Nonostante siano coinvolti **due** vettori, occore **un solo ciclo** `for`, e **un solo indice** per accedere agli elementi di entrambi i vettori
- Assolutamente non tentare di fare la copia in una sola istruzione!

```
w = v ;
w[] = v[] ;
w[N] = v[N] ;
w[1,N] = v[1,N] ;
```

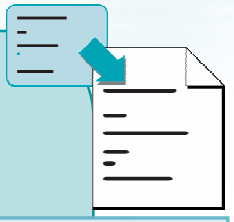
Operazioni elementari sui vettori

Ricerca di un elemento

Ricerca di un elemento

- Dato un valore numerico, verificare
  - se **almeno uno** degli elementi del vettore è uguale al valore numerico
  - in caso affermativo, dire dove si trova
  - in caso negativo, dire che non esiste
- Si tratta di una classica istanza del problema di "ricerca di esistenza"

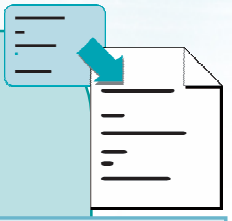
Ricerca di un elemento (1/3)



vettori.c

```
int dato ; /* dato da ricercare */
int trovato ; /* flag per ricerca */
int pos ;     /* posizione elemento */

...

printf("Elemento da ricercare? ");
scanf("%d", &dato) ;
```

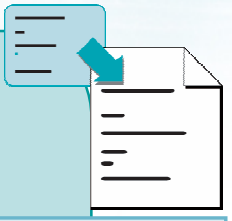
Ricerca di un elemento (2/3)



vettori.c

```
trovato = 0 ;
pos = -1 ;

for( i=0 ; i<N ; i++ )
{
    if( v[i] == dato )
    {
        trovato = 1 ;
        pos = i ;
    }
}
```

Ricerca di un elemento (3/3)



vettori.c

```
if( trovato==1 )
{
    printf("Elemento trovato "
       "alla posizione %d\n", pos+1) ;
}
else
{
    printf("Elemento non trovato\n");
}
```

Varianti

- Altri tipi di ricerche
  - Contare quante volte è presente l'elemento cercato
  - Cercare se esiste almeno un elemento maggiore (o minore) del valore specificato
  - Cercare se esiste un elemento approssimativamente uguale a quello specificato
  - ...

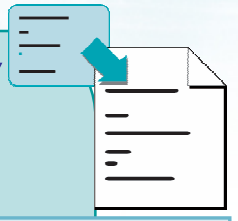
Operazioni elementari sui vettori

Ricerca del massimo o minimo

Ricerca del massimo

- Dato un vettore (di interi o reali), determinare
  - quale sia l'elemento di valore massimo
  - quale sia la posizione in cui si trova tale elemento
- Conviene applicare la stessa tecnica per l'identificazione del massimo già vista in precedenza
  - Conviene inizializzare il max al valore del primo elemento

Ricerca del massimo (1/2)



vettori.c

```
float max ;  /* valore del massimo */
int posmax ; /* posizione del max */


...
max = r[0] ;
posmax = 0 ;

for( i=1 ; i<N ; i++ )
{
     if( r[i]>max )
     {
          max = r[i] ;
          posmax = i ;
     }
}
```

Ricerca del massimo (2/2)

vettori.c

```
printf("Il max vale %f e si ", max) ;
printf("trova in posiz. %d\n", posmax) ;
```

Operazioni elementari sui vettori

Vettori ad occupazione variabile

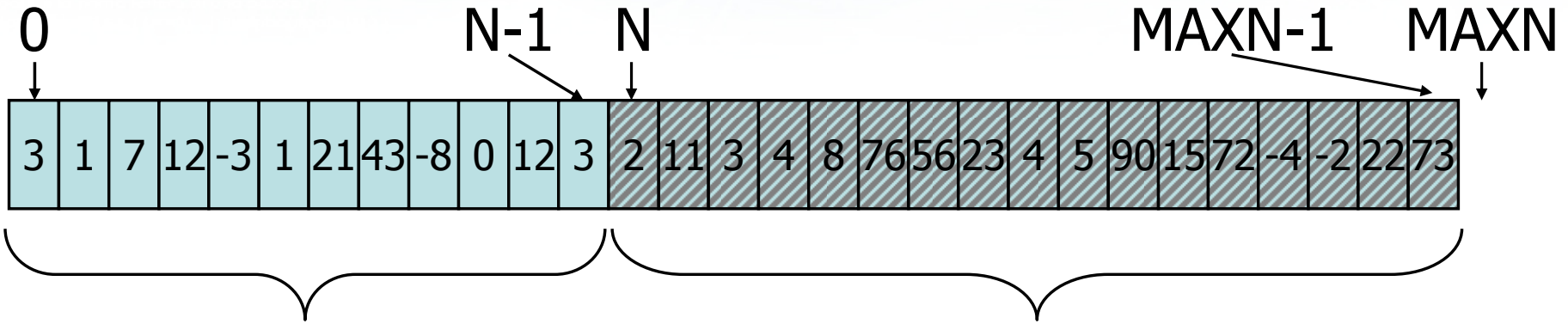
Occupazione variabile

- La principale limitazione dei vettori è la loro dimensione fissa, definita come costante al tempo di compilazione del programma
- Molto spesso non si conosce l'effettivo numero di elementi necessari fino a quando il programma non andrà in esecuzione
- Occorre identificare delle tecniche che ci permettano di lavorare con vettori di dimensione fissa, ma occupazione variabile

Tecnica adottata

- Dichiarare un vettore di dimensione sufficientemente ampia da contenere il massimo numero di elementi nel caso peggiore
  - Esempio: MAXN
- La parte iniziale del vettore sarà occupata dagli elementi, la parte finale rimarrà inutilizzata
- Dichiarare una variabile che tenga traccia dell'effettiva occupazione del vettore
  - Esempio: N

Tecnica adottata



Elementi effettivamente utilizzati

Elementi inutilizzati

Esempio

```
/* dimensione massima */
const int MAXN = 100 ;

int v[MAXN] ; /* vettore di dim. max. */

int N ; /* occupazione effettiva
           del vettore */

...

N = 0 ; /* inizialmente “vuoto” */
```

Regole di utilizzo

- All'inizio del programma si inizializza N al numero effettivo di elementi
  - Esempio: `scanf("%d", &N);`
- Verificare sempre che `N<=MAXN`
- Durante l'esecuzione, utilizzare sempre `N`, e mai `MAXN`
  - Esempio: `for(i=0; i<N; i++)`
- Gli elementi da `v[N]` a `v[MAXN-1]` vengono ignorati (costituiscono memoria "sprecata")

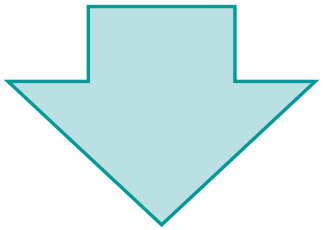
Crescita del vettore

- Un vettore ad occupazione variabile può facilmente crescere, aggiungendo elementi "in coda"

```
v[N] = nuovo_elemento ;
N++ ;
```

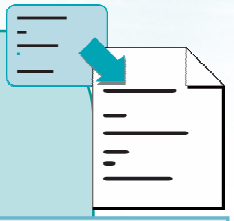
Esempio

- Acquisire da tastiera una serie di numeri reali, e memorizzarli in un vettore.
- La serie di numeri è terminata da un valore uguale a 0



- Il valore di N non è noto all'inizio del programma, ma viene aggiornato via via che si leggono gli elementi

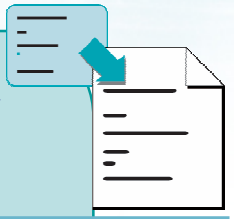
Soluzione (1/3)



leggi0.c

```
const int MAXN = 100 ;

float v[MAXN] ;
float dato ;

int N ;
int i ;

printf("Inserisci gli elementi\n") ;
printf("(per terminare 0)\n") ;
```

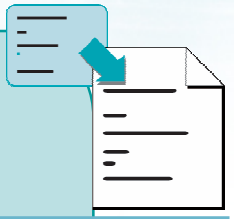
Soluzione (2/3)



leggi0.c

```
N = 0 ; /* vettore inizialm. vuoto */

/* leggi il primo dato */
i = 0 ;
printf("Elemento %d: ", i+1) ;
scanf("%f", &dato) ;
i++ ;

/* aggiungi al vettore */
if( dato != 0.0 )
{
    v[N] = dato ;
    N++ ;
}
```

Soluzione (3/3)



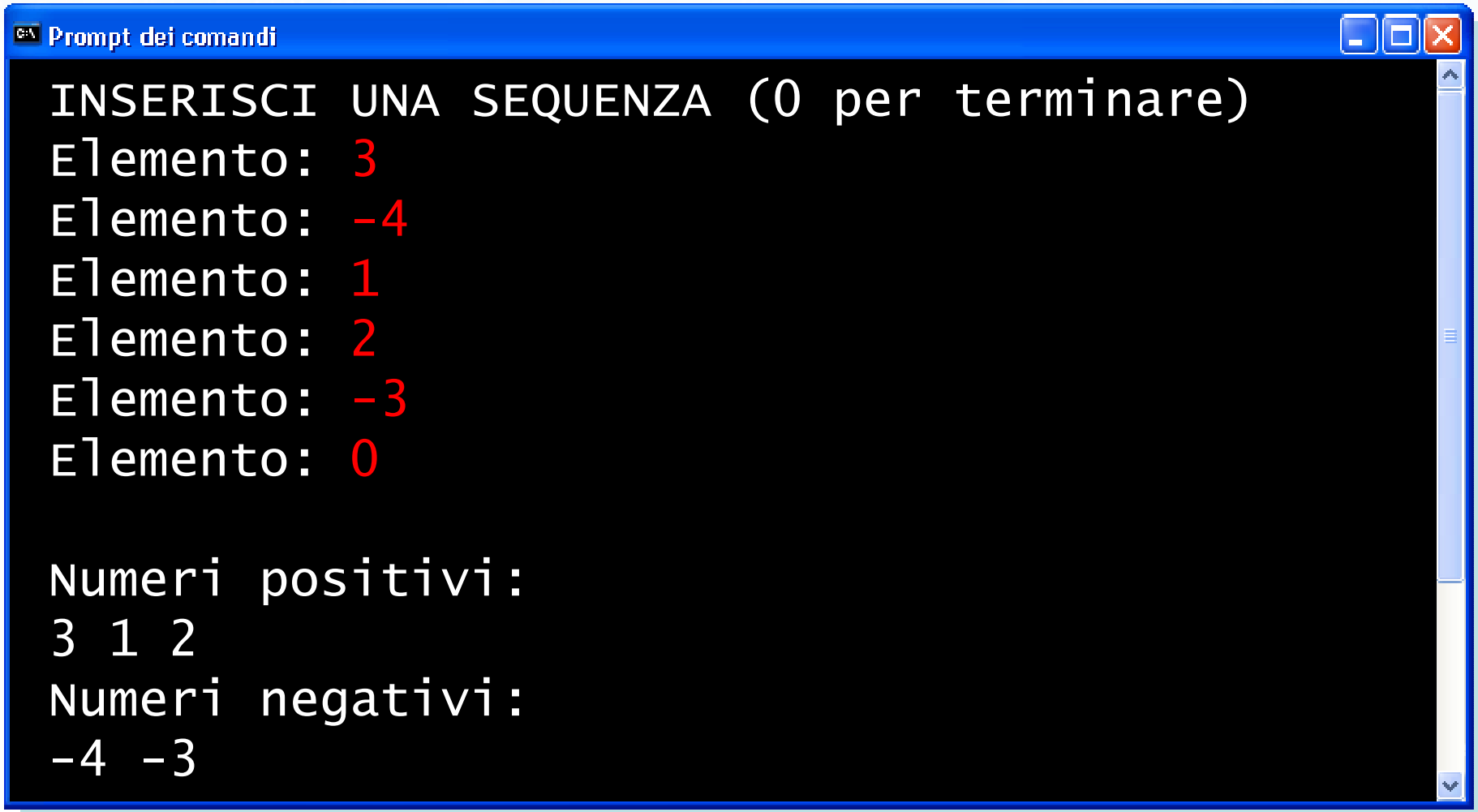
leggi0.c

```
while(dato!= 0.0)
{
     /* leggi il dato successivo */
     printf("Elemento %d: ", i+1) ;
     scanf("%f", &dato) ;
     i++ ;

     /* aggiungi al vettore */
     if( dato != 0.0 )
     {
          v[N] = dato ;
          N++ ;
     }
}
```

Esercizio "Positivi e Negativi"

- Si realizzi un programma che legga da tastiera una sequenza di numeri interi (terminata da 0), e che successivamente stampi
  - tutti i numeri positivi presenti nella sequenza, nello stesso ordine
  - tutti i numeri negativi presenti nella sequenza, nello stesso ordine

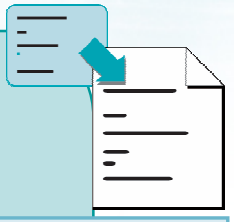
Analisi


Prompt dei comandi
INSERISCI UNA SEQUENZA (0 per terminare)
Elemento: 3
Elemento: -4
Elemento: 1
Elemento: 2
Elemento: -3
Elemento: 0
Numeri positivi:
3 1 2
Numeri negativi:
-4 -3

Approccio risolutivo

- Definiamo 3 vettori ad occupazione variabile:
  - `seq`, di occupazione `N`, che memorizza la sequenza iniziale
  - `pos`, di occupazione `Np`, che memorizza i soli elementi positivi
  - `neg`, di occupazione `Nn`, che memorizza i soli elementi negativi
- Il programma inizialmente acquisirà da tastiera il vettore `seq`, in seguito trascriverà ciascun elemento nel vettore più opportuno

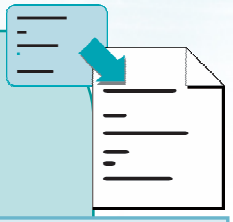
Soluzione (1/4)



posneg.c

```
const int MAXN = 100 ;

int seq[MAXN] ;
int pos[MAXN] ;
int neg[MAXN] ;

int N, Np, Nn ;

int i ;
int dato ;

/* vettori inizialmente vuoti */
N = 0 ;
Np = 0 ;
Nn = 0 ;
```

Soluzione (2/4)

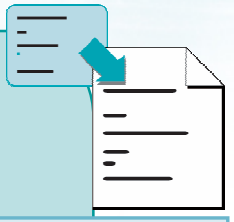


posneg.c

```
/* LETTURA SEQUENZA INIZIALE */

/* vedi esempio precedente ... */
```

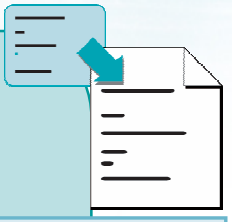
Soluzione (3/4)

```
for( i=0 ; i<N ; i++ )
{
    if(seq[i] > 0)
    {
        /* positivo => in pos[] */
        pos[Np] = seq[i] ;
        Np++ ;
    }
    else
    {
        /* negativo => in neg[] */
        neg[Nn] = seq[i] ;
        Nn++ ;
    }
}
```



posneg.c

Soluzione (4/4)



posneg.c

```
printf("Numeri positivi:\n") ;
for(i=0; i<Np; i++)
     printf("%d ", pos[i]) ;
printf("\n");

printf("Numeri negativi:\n") ;
for(i=0; i<Nn; i++)
     printf("%d ", neg[i]) ;
printf("\n");
```

Vettori

Esercizi guidati sui vettori

Esercizi guidati sui vettori

- Esercizio “Elementi comuni”
- Esercizio “Ricerca duplicati”
- Esercizio “Sottosequenza”
- Esercizio “Poligono”

Esercizi guidati sui vettori

Esercizio "Elementi comuni"

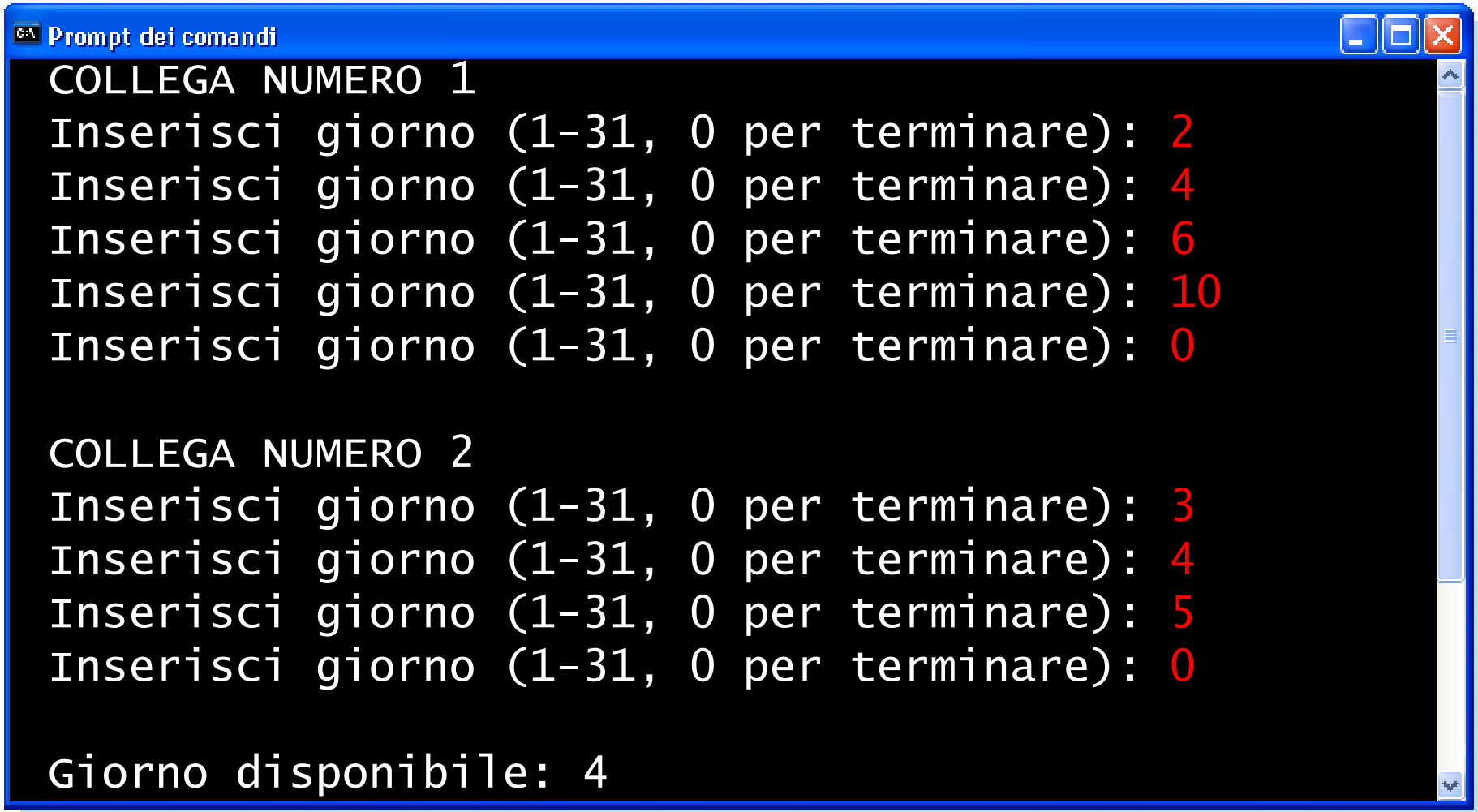
Esercizio “Elementi comuni” (1/2)

- Due colleghi intendono fissare una riunione, pertanto devono identificare dei giorni nei quali sono entrambi liberi da impegni. A tale scopo, essi realizzano un programma C che permetta a ciascuno di immettere le proprie disponibilità, e che identifichi i giorni nei quali entrambi sono liberi

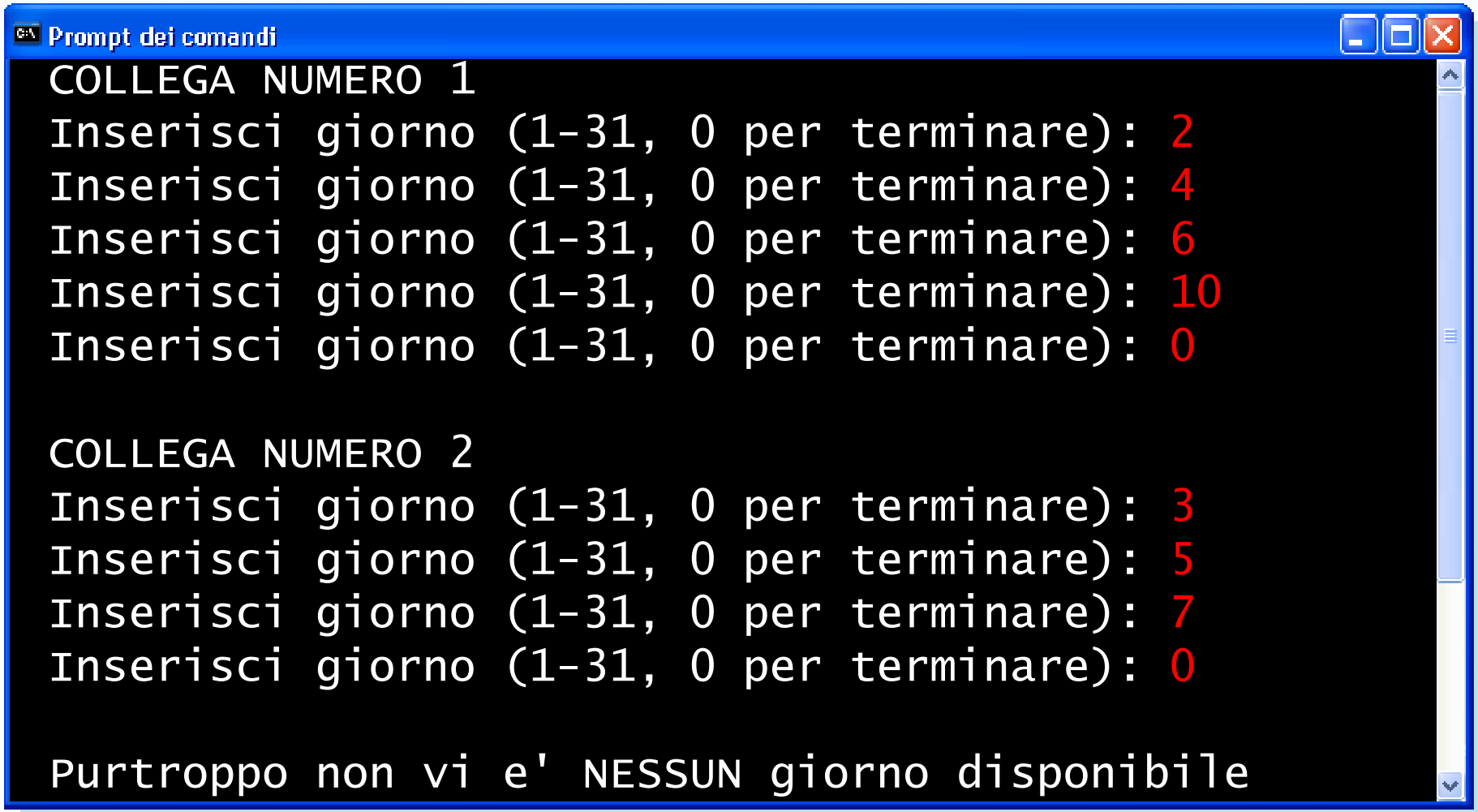
Esercizio "Elementi comuni" (2/2)

- In particolare, in una prima fase il programma acquisisce le disponibilità dei due colleghi
  - Per ciascun collega il programma acquisisce un elenco di numeri interi (supponiamo compresi tra 1 e 31), che indicano i giorni del mese in cui essi sono disponibili. L'immissione dei dati termina inserendo 0.
- Nella seconda fase, il programma identificherà i giorni in cui entrambi i colleghi sono disponibili, e li stamperà a video

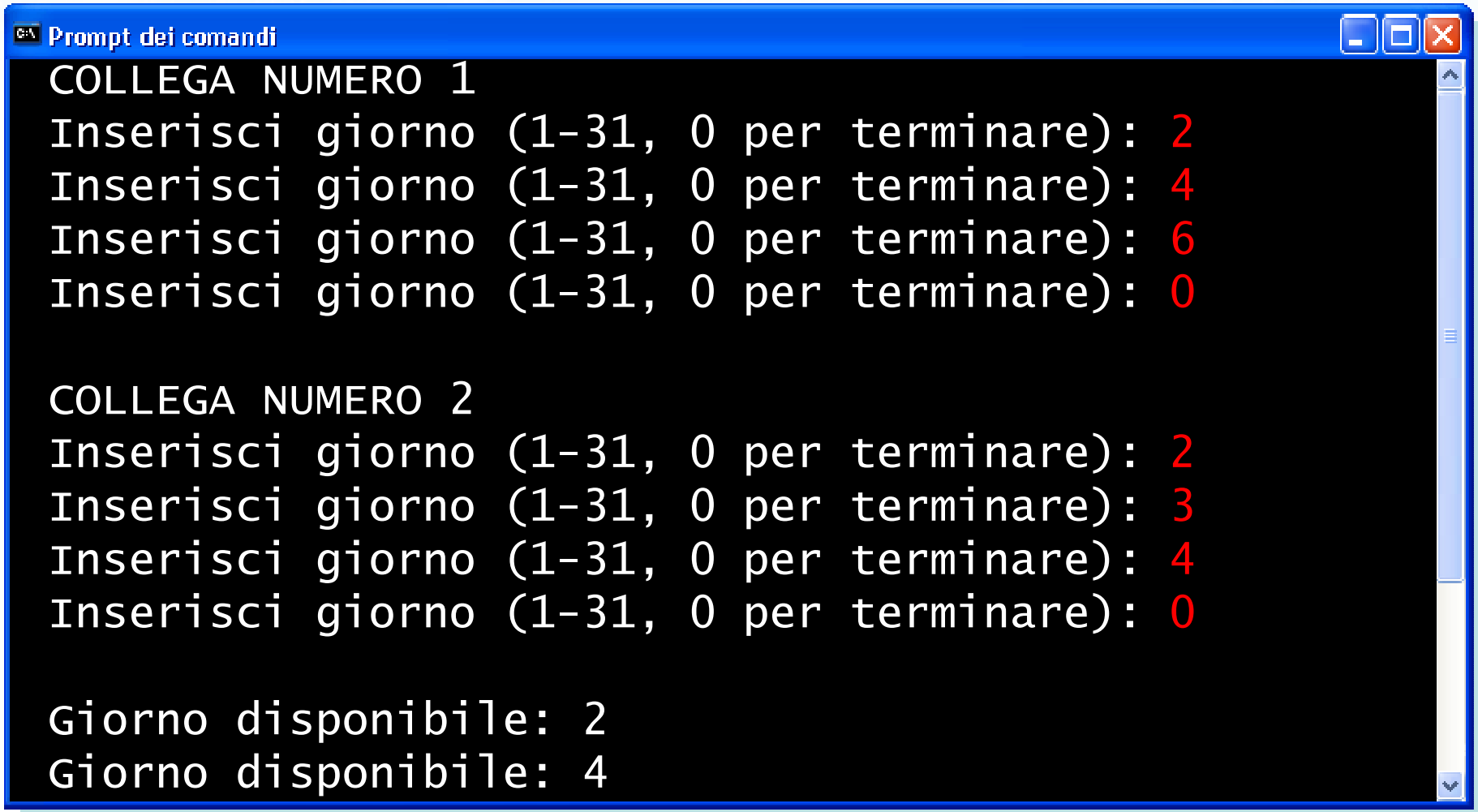
Analisi (1/3)


Prompt dei comandi
COLLEGA NUMERO 1
Inserisci giorno (1-31, 0 per terminare): 2
Inserisci giorno (1-31, 0 per terminare): 4
Inserisci giorno (1-31, 0 per terminare): 6
Inserisci giorno (1-31, 0 per terminare): 10
Inserisci giorno (1-31, 0 per terminare): 0
COLLEGA NUMERO 2
Inserisci giorno (1-31, 0 per terminare): 3
Inserisci giorno (1-31, 0 per terminare): 4
Inserisci giorno (1-31, 0 per terminare): 5
Inserisci giorno (1-31, 0 per terminare): 0
Giorno disponibile: 4

Analisi (2/3)


Prompt dei comandi
COLLEGA NUMERO 1
Inserisci giorno (1-31, 0 per terminare): 2
Inserisci giorno (1-31, 0 per terminare): 4
Inserisci giorno (1-31, 0 per terminare): 6
Inserisci giorno (1-31, 0 per terminare): 10
Inserisci giorno (1-31, 0 per terminare): 0
COLLEGA NUMERO 2
Inserisci giorno (1-31, 0 per terminare): 3
Inserisci giorno (1-31, 0 per terminare): 5
Inserisci giorno (1-31, 0 per terminare): 7
Inserisci giorno (1-31, 0 per terminare): 0
Purtroppo non vi e' NESSUN giorno disponibile

Analisi (3/3)

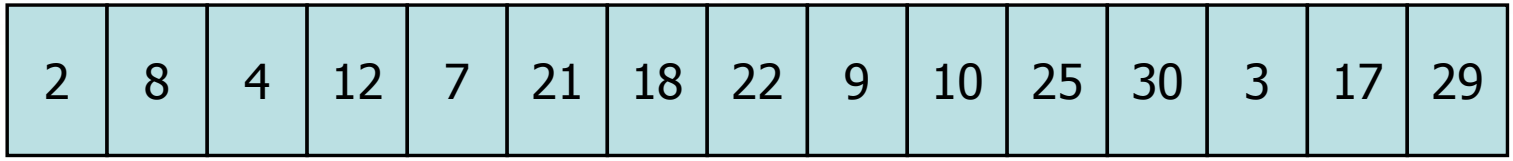

Prompt dei comandi
COLLEGA NUMERO 1
Inserisci giorno (1-31, 0 per terminare): 2
Inserisci giorno (1-31, 0 per terminare): 4
Inserisci giorno (1-31, 0 per terminare): 6
Inserisci giorno (1-31, 0 per terminare): 0
COLLEGA NUMERO 2
Inserisci giorno (1-31, 0 per terminare): 2
Inserisci giorno (1-31, 0 per terminare): 3
Inserisci giorno (1-31, 0 per terminare): 4
Inserisci giorno (1-31, 0 per terminare): 0
Giorno disponibile: 2
Giorno disponibile: 4

Algoritmo

- Acquisisci le disponibilità del collega 1
  - Vettore `giorni1[]` di `N1` elementi
- Acquisisci le disponibilità del collega 2
  - Vettore `giorni2[]` di `N2` elementi
- Verifica se vi sono elementi di `giorni1[]` che siano anche elementi di `giorni2[]`
  - Se sì, stampa tali elementi
  - Se no, stampa un messaggio

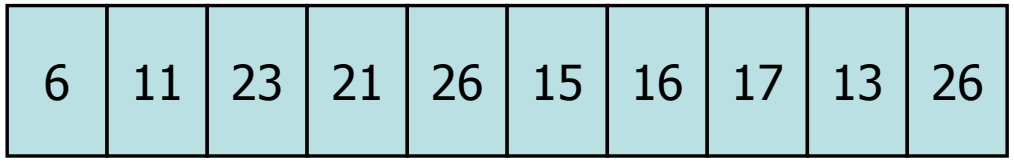
Ricerca elementi comuni

giorni1



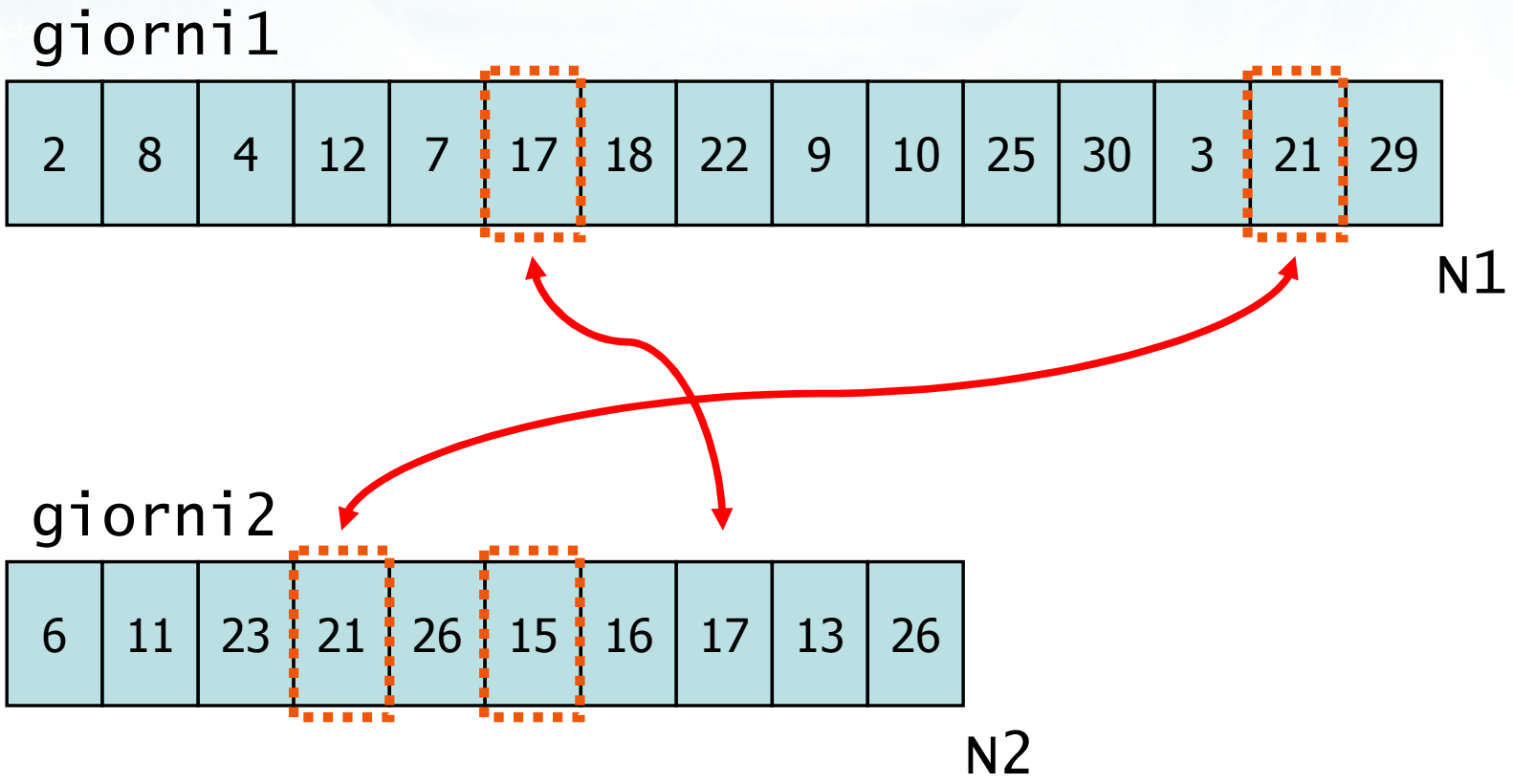
N1

giorni2

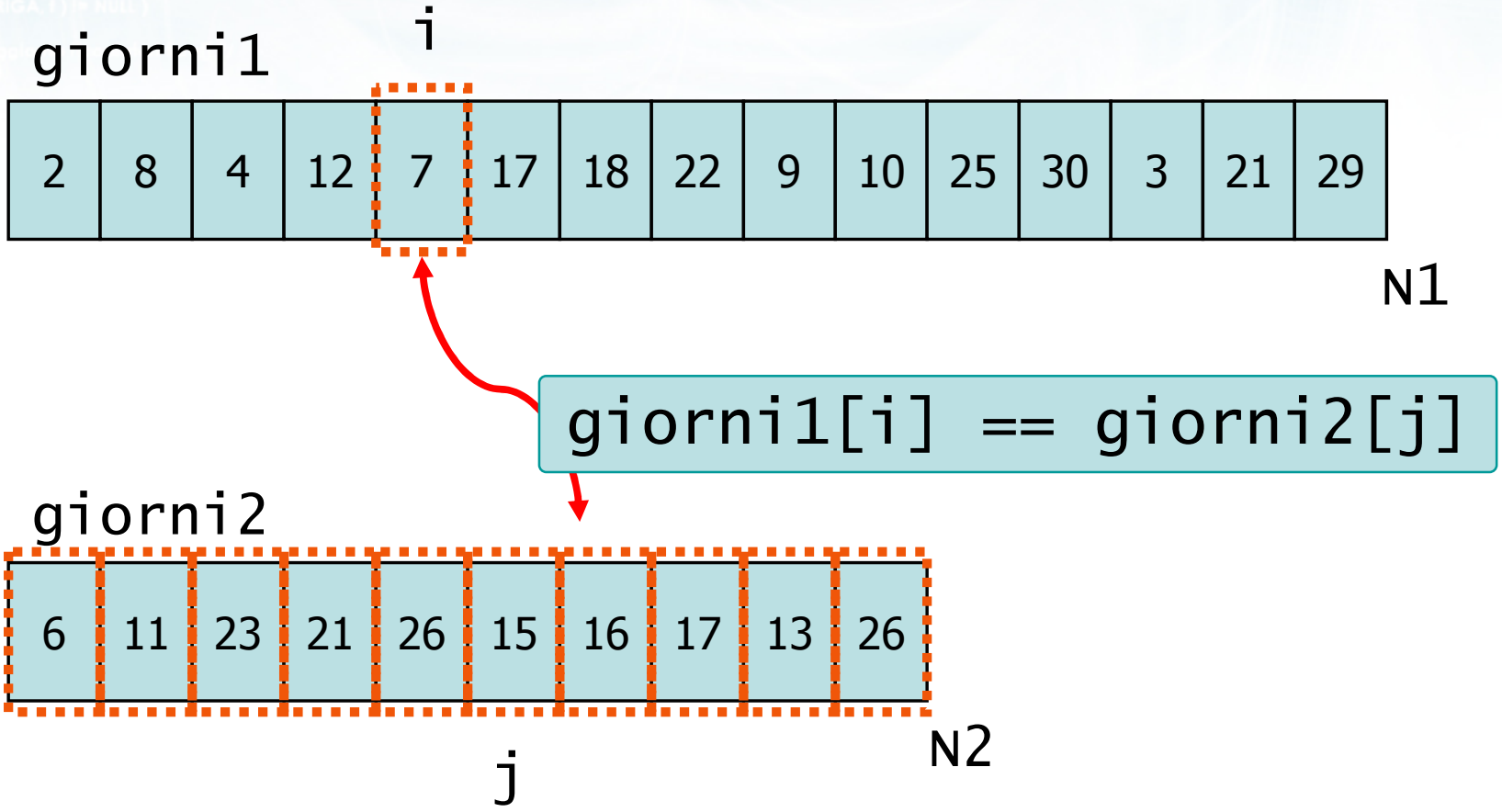


N2

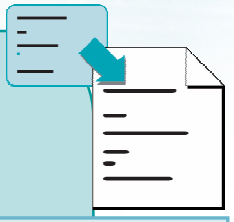
Ricerca elementi comuni



Ricerca elementi comuni



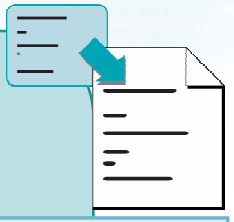
Soluzione (1/5)



riunione.c

```
const int MAXN = 100 ;

int N1, N2 ;
int giorni1[MAXN] ; /* giorni collega 1 */
int giorni2[MAXN] ; /* giorni collega 2 */

int giorno ;
int i, j ;
int trovato ;
  /* flag: giorni1[i] in giorni2[]? */
int fallito ;
  /* flag: trovato almeno un giorno? */
```

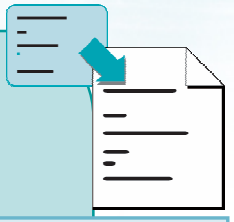
Soluzione (2/5)



riunione.c

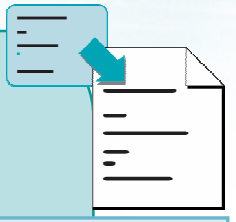
```
/* DISPONIBILITA' COLLEGA 1 */
printf("COLLEGA NUMERO 1\n");
N1 = 0 ;
printf("Inserisci giorno (1-31): ");
scanf("%d", &giorno) ;

while( giorno != 0 )
{
    giorni1[N1] = giorno ;
    N1++ ;

    printf("Inserisci giorno (1-31): ");
    scanf("%d", &giorno) ;
}
printf("Collega 1 ha inserito %d giorni\n",
    N1);
```

Soluzione (3/5)




```
/* DISPONIBILITA' COLLEGA 2 */
printf("COLLEGA NUMERO 2\n");
N2 = 0 ;
printf("Inserisci giorno (1-31): ");
scanf("%d", &giorno) ;

while( giorno != 0 )
{
    giorni2[N2] = giorno ;
    N2++ ;

    printf("Inserisci giorno (1-31): ");
    scanf("%d", &giorno) ;
}
printf("Collega 2 ha inserito %d giorni\n",
    N2);
```

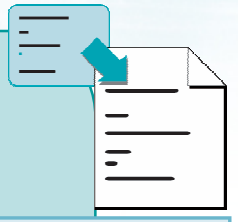
Soluzione (4/5)



riunione.c

```
/* RICERCA DEGLI ELEMENTI COMUNI */
fallito = 1 ;
/* Per ogni giorno del collega 1... */
for( i=0 ; i<N1; i++ )
{
  /* ...verifica se è disponibile
        il collega 2... */

  /* ...in caso affermativo stampalo */
  if( trovato == 1 )
  {
    printf("Giorno disponibile: %d\n",
      giorni1[i]) ;
    fallito = 0 ;
  }
}
```

Soluzione (4/5)

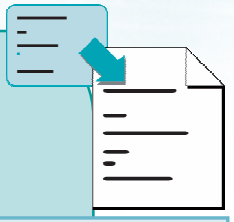


riunione.c

```
/* RICERCA DEGLI ELEMENTI COMUNI */
fallito = 1 ;
/* Per ogni giorno del collega 1... */
for( i=0 ; i<N1; i++ )
{
  /* ...verifica se è disponibile
        il collega 2... */

  /* ...in caso affermativo stampalo */
  if( tr
  {
    prin
      gi
    fall
  }
}
```

```
trovato = 0 ;
for( j=0; j<N2; j++ )
{
  if( giorni2[j] == giorni1[i] )
    trovato = 1 ;
}
```

Soluzione (5/5)



riunione.c

```
/* Se non ne ho trovato nessuno,
   stampo un messaggio */
if(fallito==1)
  printf("NESSUN giorno disponibile\n");
```

Esercizi guidati sui vettori

Esercizio "Ricerca duplicati"

Esercizio "Ricerca duplicati" (1/2)

- La società organizzatrice di un concerto vuole verificare che non ci siano biglietti falsi. A tale scopo, realizza un programma in linguaggio C che acquisisce i numeri di serie dei biglietti e verifica che non vi siano numeri duplicati

Esercizio "Ricerca duplicati" (2/2)

- In particolare, il programma acquisisce innanzitutto il numero di biglietti venduti, N, ed in seguito acquisisce i numeri di serie degli N biglietti
- Al termine dell'acquisizione, il programma stamperà "Tutto regolare" se non si sono riscontrati duplicati, altrimenti stamperà il numero di serie dei biglietti duplicati

Analisi (1/2)


Prompt dei comandi
RICERCA BIGLIETTI DUPLICATI
Numero totale di biglietti: 5
Numero di serie del biglietto 1: 1234
Numero di serie del biglietto 2: 4321
Numero di serie del biglietto 3: 1423
Numero di serie del biglietto 4: 1242
Numero di serie del biglietto 5: 3321
Tutto regolare

Analisi (2/2)


Prompt dei comandi
RICERCA BIGLIETTI DUPLICATI
Numero totale di biglietti: 5
Numero di serie del biglietto 1: 1234
Numero di serie del biglietto 2: 4321
Numero di serie del biglietto 3: 1234
Numero di serie del biglietto 4: 1242
Numero di serie del biglietto 5: 3321
ATTENZIONE: biglietto 1234 duplicato!

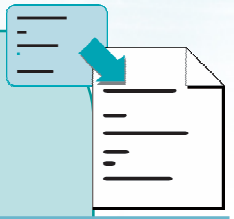
Algoritmo

- Acquisizione del valore di N
- Lettura dei numeri di serie
  - Utilizziamo un vettore: `int serie[MAXN]`
- **Ricerca dei duplicati**
- Stampa dei messaggi finali

Ricerca dei duplicati

- Prendi un elemento per volta
  - `elem = serie[i] ;`
- Cerca se **altri** elementi del vettore sono **uguali** a tale elemento
  - uguali $\Rightarrow$ `(elem == serie[j])`
  - altri $\Rightarrow$ `(i != j)`
- Si tratta di una ricerca di esistenza per ogni elemento considerato

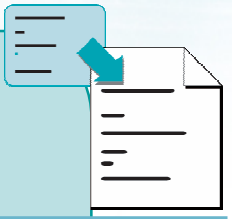
Soluzione (1/5)



biglietti.c

```
const int MAXN = 100 ;

int N ; /* num tot biglietti */
int serie[MAXN] ; /* numeri serie */

int elem ;
int i, j ;

/* flag: trovato almeno un duplic.? */
int dupl ;

/* flag per ricerca di esistenza */
int trovato ;
```

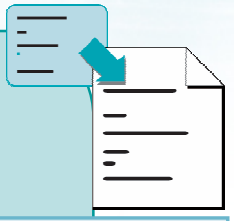
Soluzione (2/5)



biglietti.c

```
printf("RICERCA DUPLICATI\n") ;
printf("\n");

/* ACQUISIZIONE VALORE DI N */
do{

  printf("Num tot di biglietti: ") ;
  scanf("%d", &N) ;
  if (N<2 || N>MAXN)
    printf("N=%d non valido\n", N) ;

} while(N<2 || N>MAXN) ;
```

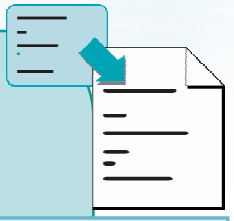
Soluzione (3/5)



biglietti.c

```
/* LETTURA DEI NUMERI DI SERIE */
for( i=0 ; i<N ; i++ )
{
  printf("Numero serie biglietto %d: ",
          i+1) ;
  scanf("%d", &serie[i]) ;
}
```

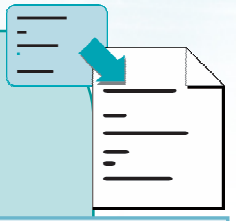
Soluzione (4/5)



biglietti.c

```
/* RICERCA DEI DUPLICATI */
dupl = 0 ;
for( i=0 ; i<N ; i++ )
{
  /* verifica se serie[i] e' duplicato */
  elem = serie[i] ;

  /* => ricerca esistenza di elem
        all'interno di serie[] */

  if(trovato == 1)
  {
    printf("ATTENZIONE: %d duplicato\n",
            elem) ;
    dupl = 1 ;
  }
}
```

Soluzione (4/5)

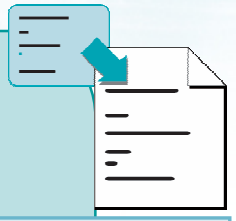


biglietti.c

```
/* RICERCA DEI DUPLICATI */
dupl = 0 ;
for( i=0 ; i<N ; i++ )
{
  /* verifica se serie[i] e' duplicato */
  elem = serie[i] ;

  /* => ricerca esistenza di elem
        all'interno di serie[] */

  if(trovato
  {
    printf("A
            el
    dupl = 1
  }
}
```

```
trovato = 0 ;
for( j=0 ; j<N ; j++ )
{
  if( (i!=j) &&
      (elem == serie[j]) )
    trovato = 1 ;
}
```

Soluzione (5/5)



biglietti.c

```
/* STAMPA DEI MESSAGGI FINALI */
if(dupl==0)
    printf("Tutto regolare\n");
```

Esercizi guidati sui vettori

Esercizio "Sottosequenza"

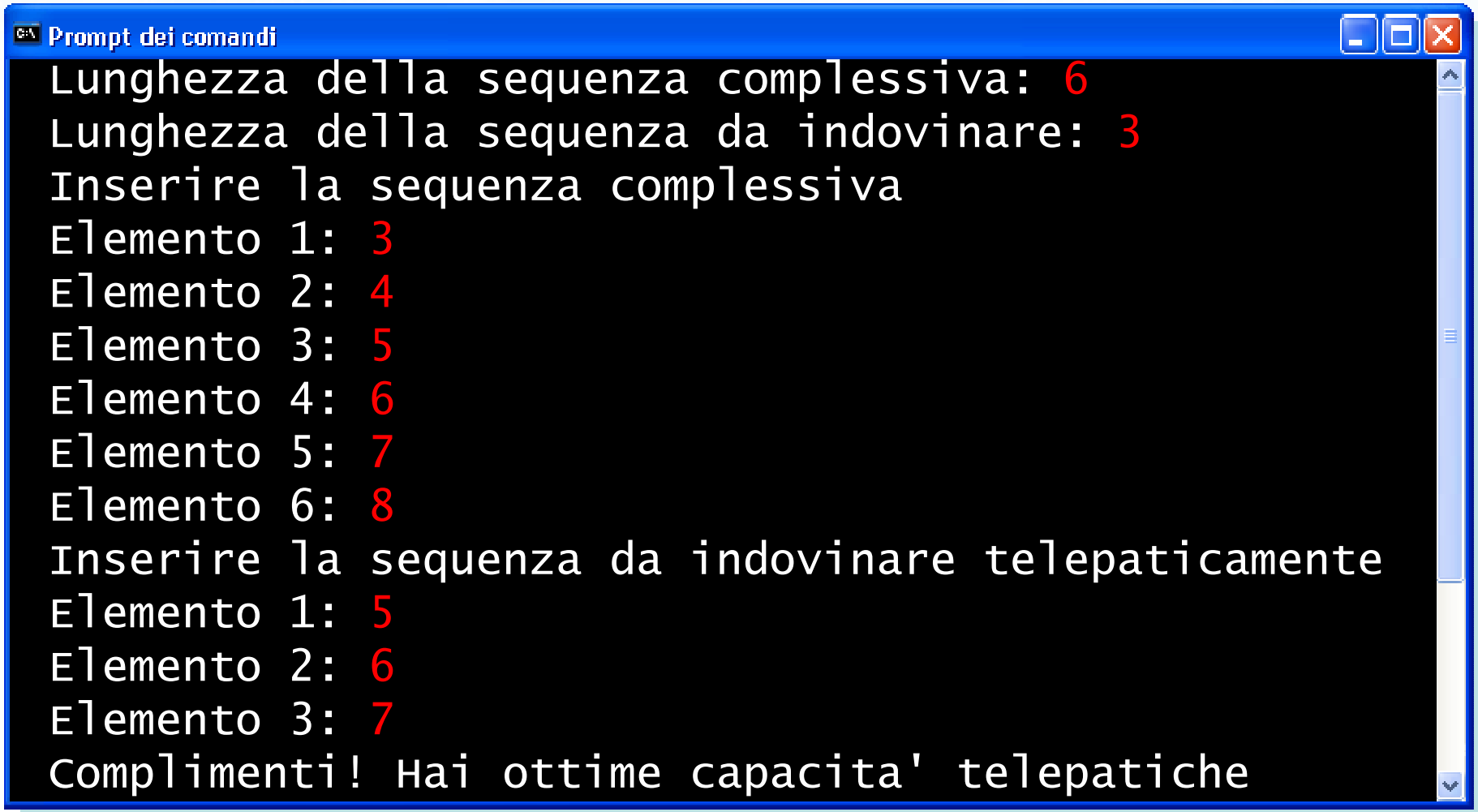
Esercizio "Sottosequenza" (1/2)

- In un esercizio di telepatia, un sensitivo scommette di essere in grado di indovinare almeno 3 numeri consecutivi, in una sequenza di 100 numeri pensati da uno spettatore
- Per garantire l'oggettività dell'esperimento, viene realizzato un programma in C per la verifica dell'avvenuta telepatia
  - Per maggior generalità, il programma viene realizzato in modo da controllare sequenze di almeno K numeri consecutivi, all'interno di sequenze di N numeri.

Esercizio "Sottosequenza" (2/2)

- In particolare, il programma acquisisce innanzitutto la sequenza di N numeri pensati dallo spettatore. Si ipotizza che tali numeri siano interi positivi, compresi tra 1 e 10000
- In seguito, il programma acquisisce dal sensitivo una sequenza di K numeri
- Il programma verifica se esiste, nella sequenza di N numeri, una sottosequenza di K numeri esattamente uguale a quella inserita dal sensitivo
- I valori di N e K sono introdotti dall'utente all'inizio del programma

Analisi


Prompt dei comandi
Lunghezza della sequenza complessiva: 6
Lunghezza della sequenza da indovinare: 3
Inserire la sequenza complessiva
Elemento 1: 3
Elemento 2: 4
Elemento 3: 5
Elemento 4: 6
Elemento 5: 7
Elemento 6: 8
Inserire la sequenza da indovinare telepaticamente
Elemento 1: 5
Elemento 2: 6
Elemento 3: 7
Complimenti! Hai ottime capacita' telepatiche

Algoritmo (1/2)

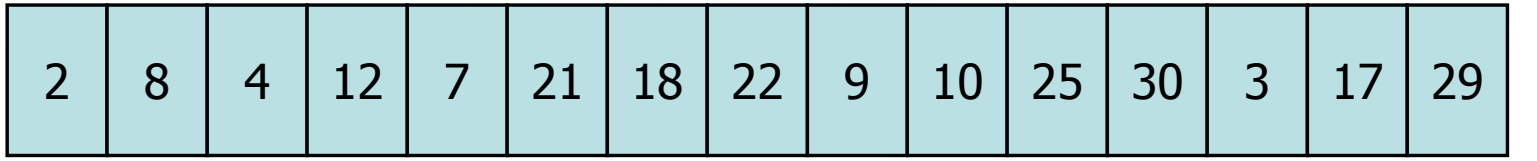
- Chiamiamo
  - `seq[]` la sequenza di N elementi
  - `tele[]` la sottosequenza di K elementi (K<N)

Algoritmo (2/2)

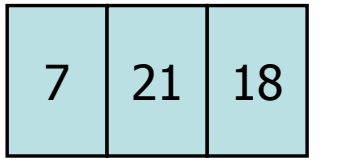
- Verifichiamo se
  - i primi K elementi di `seq[]` sono uguali ai K elementi di `tele[]`
  - i K elementi di `seq[]` con indice da 1 a K sono uguali ai K elementi di `tele[]`
  - i K elementi di `seq[]` con indice da 2 a K+1 sono uguali ai K elementi di `tele[]`
  - i K elementi di `seq[]` con indice da 3 a K+2 sono uguali ai K elementi di `tele[]`
  - ...

Sottosequenze

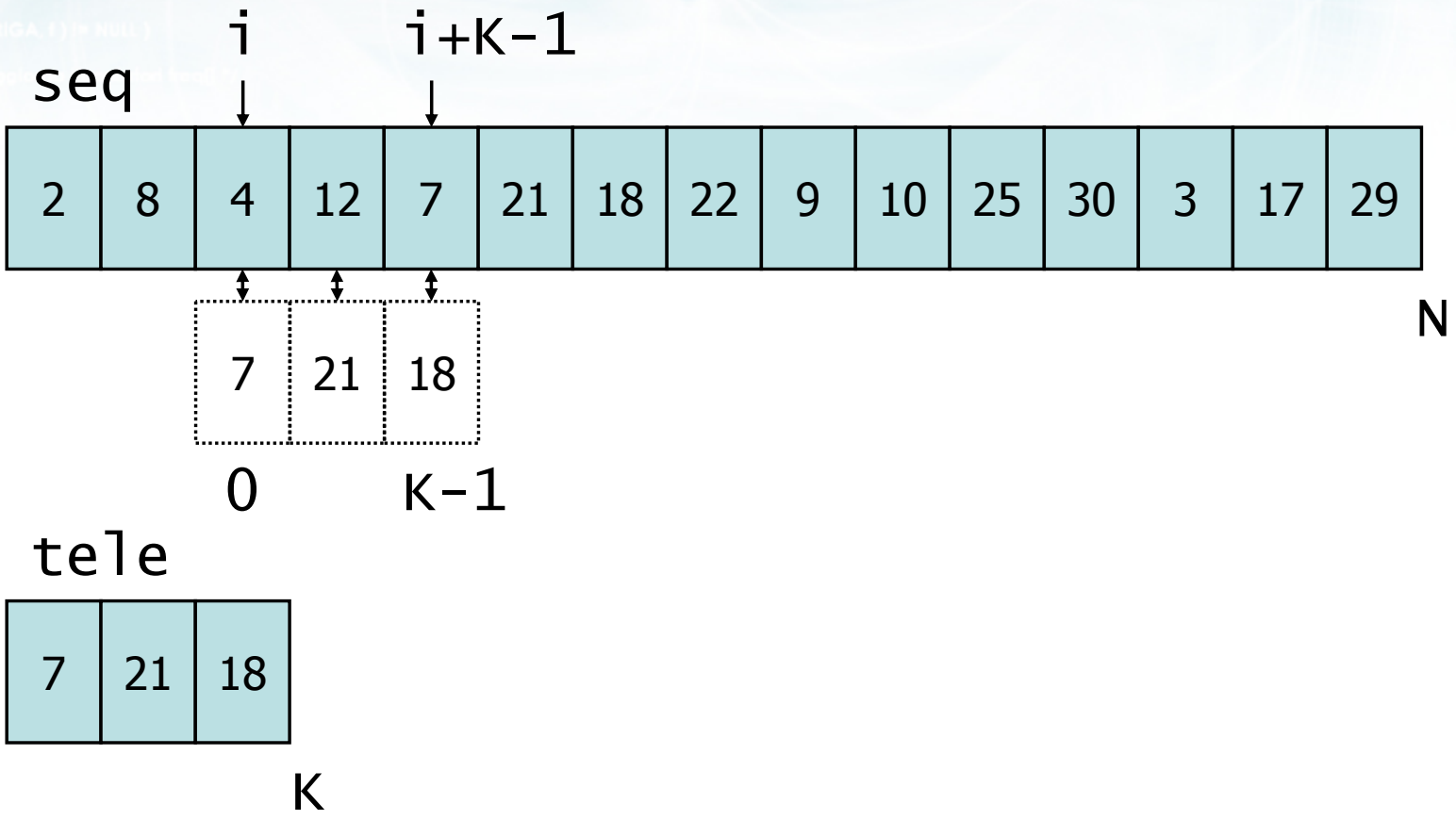
seq



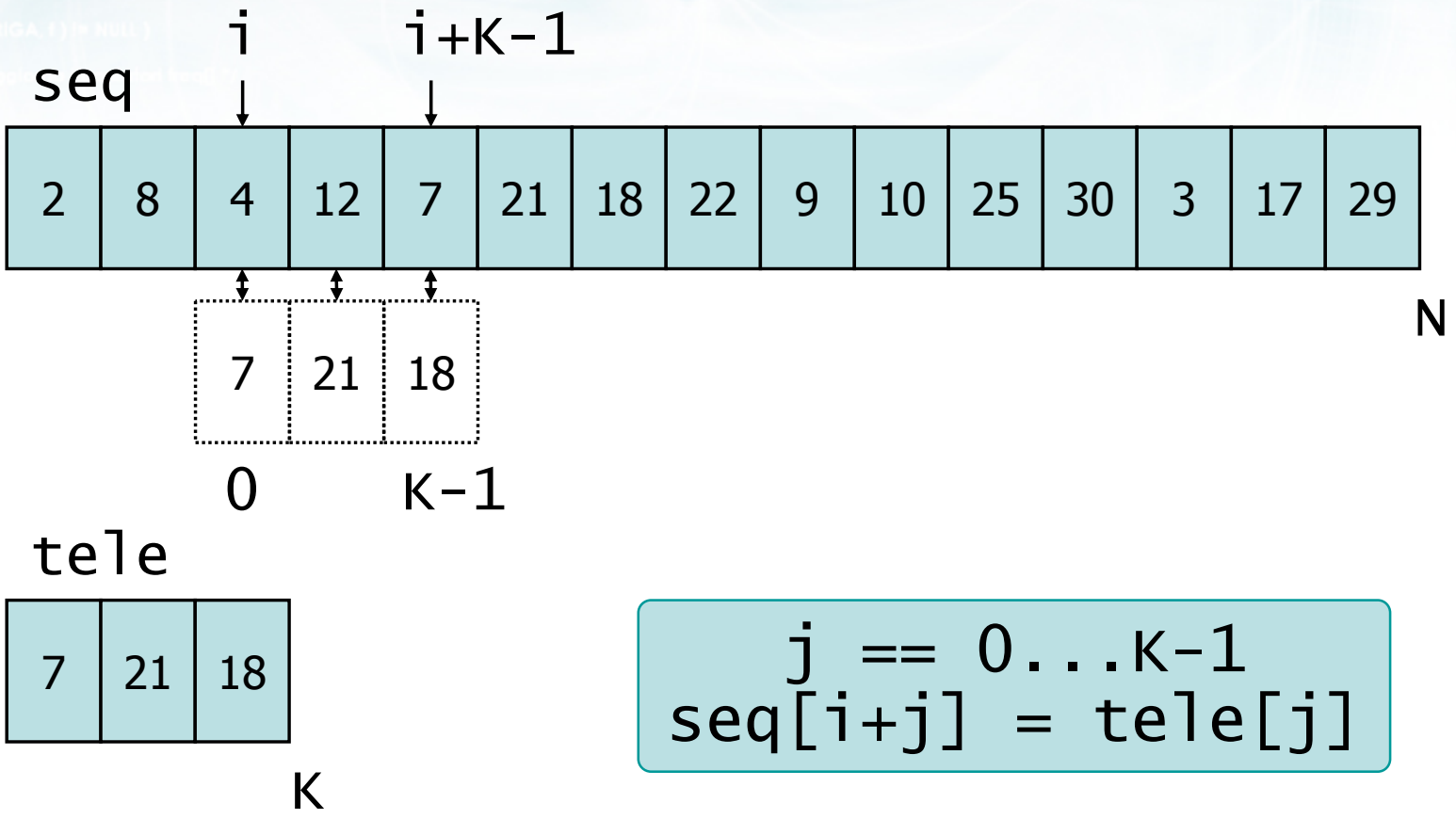
tele



Sottosequenze

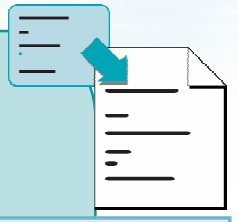


Sottosequenze




```
j == 0...K-1
seq[i+j] = tele[j]
```

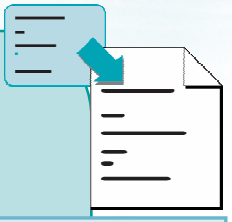
Soluzione (1/4)



sensitivo.c

```
const int MAXN = 100 ;
const int MAXK = 10 ;

int N ; /* lunghezza seq. completa */
int K ; /* lunghezza sottosequenza */
int seq[MAXN] ; /* seq. completa */
int tele[MAXK] ; /* seq. telepatica */

int i, j ;
int trovato ; /* flag: ricorda se ha
   trovato una sottosequenza uguale */
int errore ; /* flag: verifica che
   TUTTI gli elementi della
   sottosequenza siano uguali */
```

Soluzione (2/4)



sensitivo.c

```
printf("ESPERIMENTO DI TELEPATIA\n") ;
printf("\n");

/* ACQUISIZIONE LUNGHEZZA SEQUENZE */
...leggi da tastiera i valori di N e K...

/* ACQUISIZIONE SEQUENZA COMPLESSIVA */
...leggi da tastiera il vettore seq[] di N elementi...

/* ACQUISIZIONE SEQ. DA INDOVINARE */
...leggi da tastiera il vettore tele[] di K elementi...
```

Soluzione (3/4)


```
trovato = 0 ;
/* considera tutti i possibili
   punti di partenza (i) */
for( i=0; i<N-K; i++ )
{
    /* verifica se seq[] nelle
       posizioni da (i) a (i+K-1) e'
       uguale a tele[] nelle posizioni
       da (0) a (K-1) */

    /* la sottosequenza era corretta? */
    if(errore==0)
        trovato=1 ;
}
```

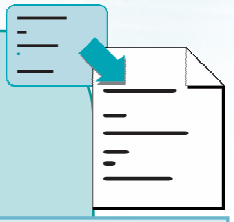
Soluzione (3/4)

sensitivo.c

```
trovato = 0 ;
/* considera tutti i possibili
   punti di partenza (i) */
for( i=0; i<N-K; i++ )
{
    /* verifica se seq[] nelle
       posizioni da (i) a (i+K-1) e'
       uguale a tele[] nelle posizioni
       da (0) a (K-1) */

    /* la                                  */
    if(err
        tr
}
```

```
errore = 0 ;
for( j=0; j<K; j++ )
{
    if( seq[i+j] != tele[j] )
        errore = 1 ;
}
```

Soluzione (4/4)



sensitivo.c

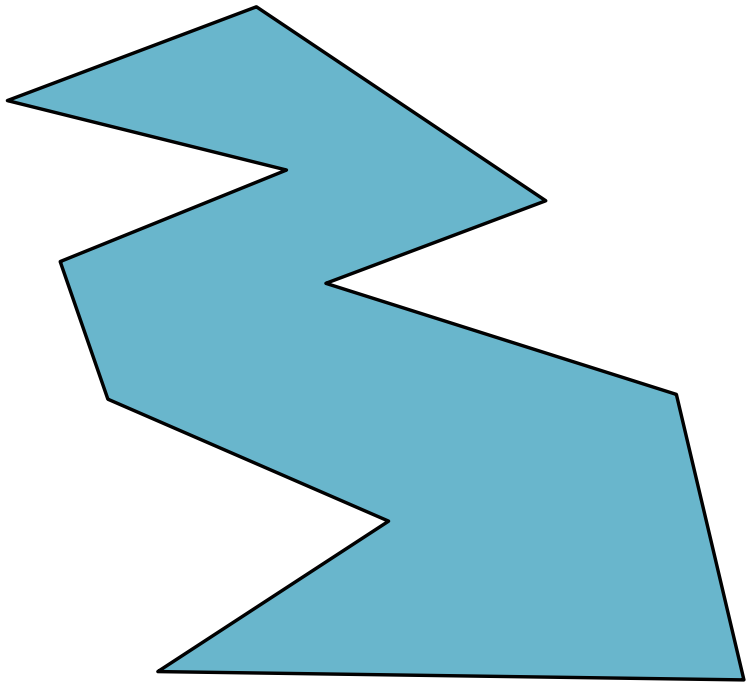
```
/* STAMPA RISULTATO DELLA VERIFICA */
if( trovato==1 )
  printf("Complimenti!\n");
else
  printf("Esperimento non riuscito\n");
```

Esercizi guidati sui vettori

Esercizio “Poligono”

Esercizio "Poligono" (1/2)

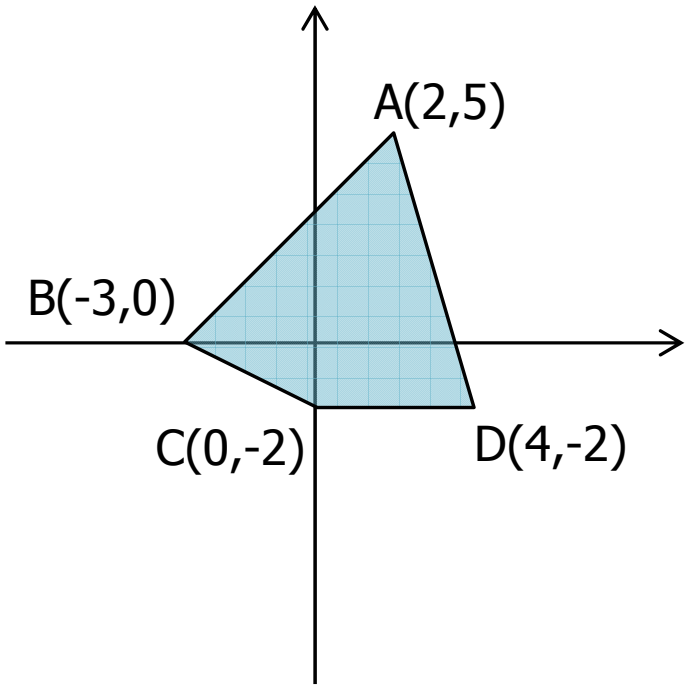
- Uno studente di geometria deve misurare il perimetro di una serie di poligoni irregolari, di cui conosce le coordinate cartesiane (x,y) dei vertici. Per far ciò realizza un programma in C



Esercizio "Poligono" (2/2)

- In particolare, il programma innanzitutto acquisisce il numero di vertici di cui è composto il poligono. Chiamiamo N tale numero
- In seguito, il programma acquisisce le N coppie (x,y) corrispondenti agli N vertici
- Il programma infine stampa la lunghezza complessiva del perimetro del poligono irregolare:
  - sommando delle lunghezze di ciascun lato
  - lato determinato dalle coordinate dei vertici

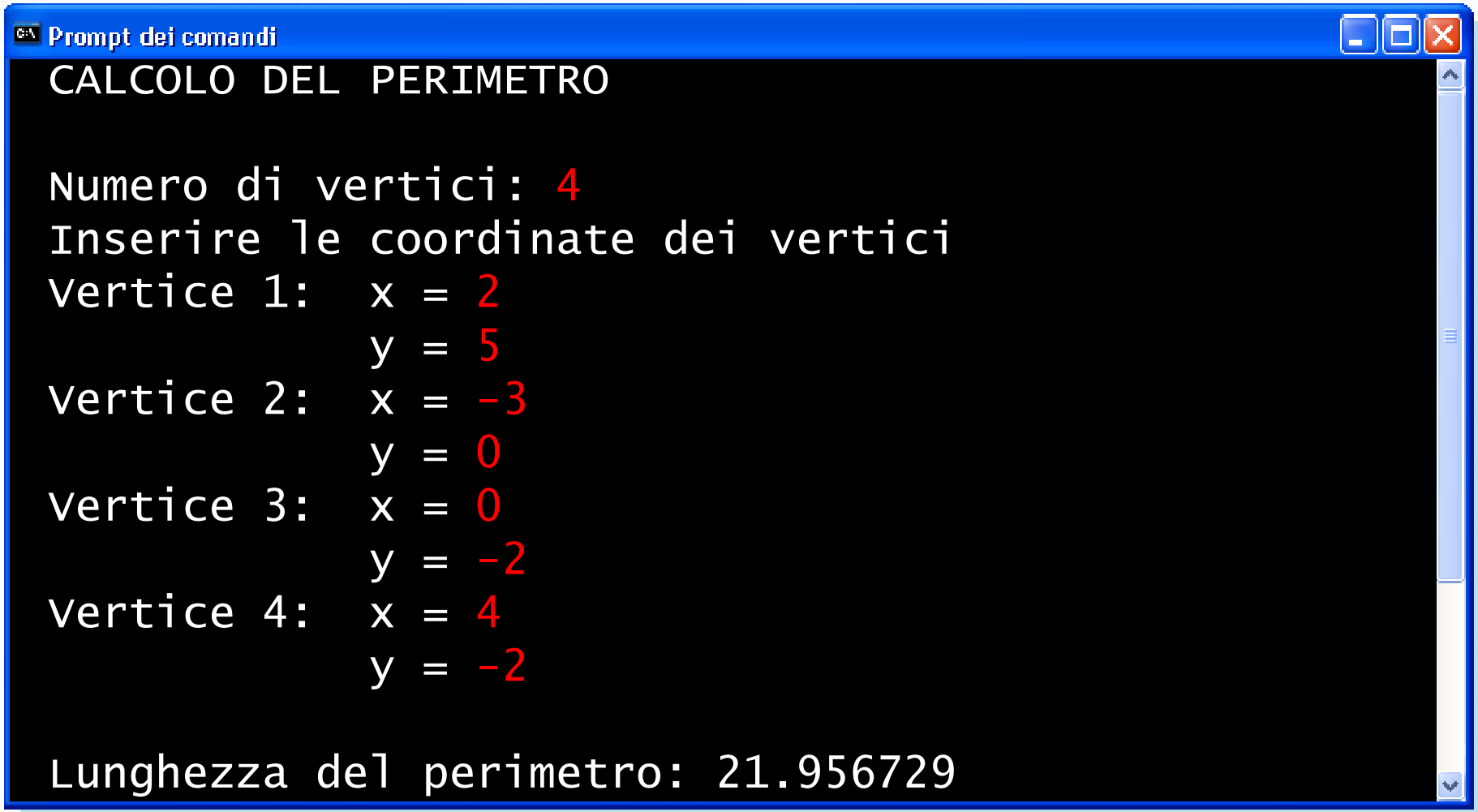
Analisi (1/2)



- AB = $\sqrt{(2-(-3))^2+(5-0)^2}$
- BC = $\sqrt{(-3-0)^2+(0-(-2))^2}$
- CD = $\sqrt{(0-4)^2+(-2-(-2))^2}$
- DA = $\sqrt{(4-2)^2+(-2-5)^2}$

- Perimetro =
  - = AB + BC + CD + DA

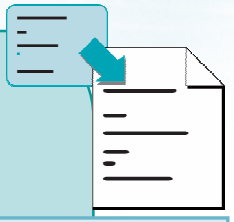
Analisi (2/2)


Prompt dei comandi
CALCOLO DEL PERIMETRO
Numero di vertici: 4
Inserire le coordinate dei vertici
Vertice 1: x = 2
y = 5
Vertice 2: x = -3
y = 0
Vertice 3: x = 0
y = -2
Vertice 4: x = 4
y = -2
Lunghezza del perimetro: 21.956729

Struttura dati

- In questo problema i dati da memorizzare non sono semplici numeri, ma coppie di numeri
- Possiamo utilizzare due vettori
  - Vettore `x[]`, contenente le ascisse dei punti
  - Vettore `y[]`, contenente le ordinate dei punti
- Esempio:
  - Punto A(2,5) $\Rightarrow$ `x[0]=2 ; y[0]=5 ;`
- In tutte le elaborazioni, i due vettori verranno usati con **uguale valore dell'indice**
  - Vettori "paralleli"

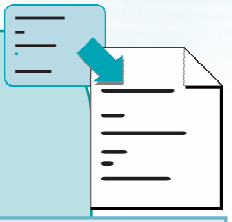
Soluzione (1/3)



poligoni.c

```
const int MAXN = 10 ;

int N ; /* numero di vertici */

/* vettori "paralleli" */
float x[MAXN] ;
float y[MAXN] ;

int i ;

float lato ;
float perimetro ;
```

Soluzione (2/3)



poligoni.c

```
/* ACQUISIZIONE NUMERO VERTICI */
...leggi da tastiera il valore di N...

/* ACQUISIZIONE COORDINATE VERTICI */
printf("Inserire coordinate\n");
for( i=0; i<N; i++ )
{
    printf("Vertice %d:  x = ", i+1) ;
    scanf("%f", &x[i]) ;
    printf("             y = ") ;
    scanf("%f", &y[i]) ;
}
```

Soluzione (3/3)

poligoni.c

```
perimetro = 0 ;

for( i=0; i<N-1; i++ )
{
  /* (x[i],y[i])-(x[i+1],y[i+1]) */
  lato = sqrt(   (x[i]-x[i+1])*(x[i]-x[i+1])
               + (y[i]-y[i+1])*(y[i]-y[i+1]) ) ;
  perimetro = perimetro + lato ;
}

/* ultimo lato:
  (x[N-1],y[N-1])-(x[0],y[0]) */
  lato = sqrt(   (x[N-1]-x[0])*(x[N-1]-x[0])
               + (y[N-1]-y[0])*(y[N-1]-y[0]) ) ;
  perimetro = perimetro + lato ;
```

Vettori

Sommario

Argomenti trattati

- La struttura dati vettoriale
- Dichiarazione di vettori in C
- Accesso agli elementi del vettore
- Vettori con occupazione variabile

Tecniche di programmazione

- Lettura e scrittura di vettori
- Ricerca di elementi
- Ricerche di duplicati, di sottosequenze, di elementi comuni, ...
- Vettori paralleli

Vettori e cicli

- Per elaborare il contenuto di un vettore sono spesso necessari dei cicli
  - Operazioni semplici: scansione del vettore per stampa, lettura, ricerca, ...
  - Operazioni complesse: possono richiedere più cicli annidati

Errore frequente

- Non è detto che ad **ogni vettore** corrisponda **un ciclo**
- Controesempio: nella ricerca di elementi duplicati, vi è un solo vettore, ma due cicli annidati
- Controesempio: nel calcolo del perimetro, vi sono due vettori, ma un solo ciclo

Errore frequente

- Al vettore **non è associato alcun indice** particolare, per scandirne gli elementi
- Lo stesso vettore può essere usato con indici diversi
- Lo stesso indice può essere applicato a vettori diversi

Suggerimenti

- Tenere separate le operazioni di lettura/scrittura dalle elaborazioni vere e proprie
- Procedere per gradi, in modalità top-down, cercando di riconoscere ove possibile le strutture note
  - ricerca di un elemento
  - verifica di esistenza
  - verifica di universalità
- Non confondere i vari "flag" utilizzati in caso di cicli annidati

Avvertenze

- La combinazione di vettori e cicli crea un **fortissimo incremento nella complessità** dei programmi realizzabili
  - Questo è il punto più ripido nella curva di apprendimento
- Prima di procedere oltre, allenarsi con molti esercizi di programmazione

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo