



Capitolo 4: Problem-solving con uso di vettori

DAL PROBLEMA AL PROGRAMMA:
INTRODUZIONE AL PROBLEM-SOLVING IN
LINGUAGGIO C



I vettori nel problem solving

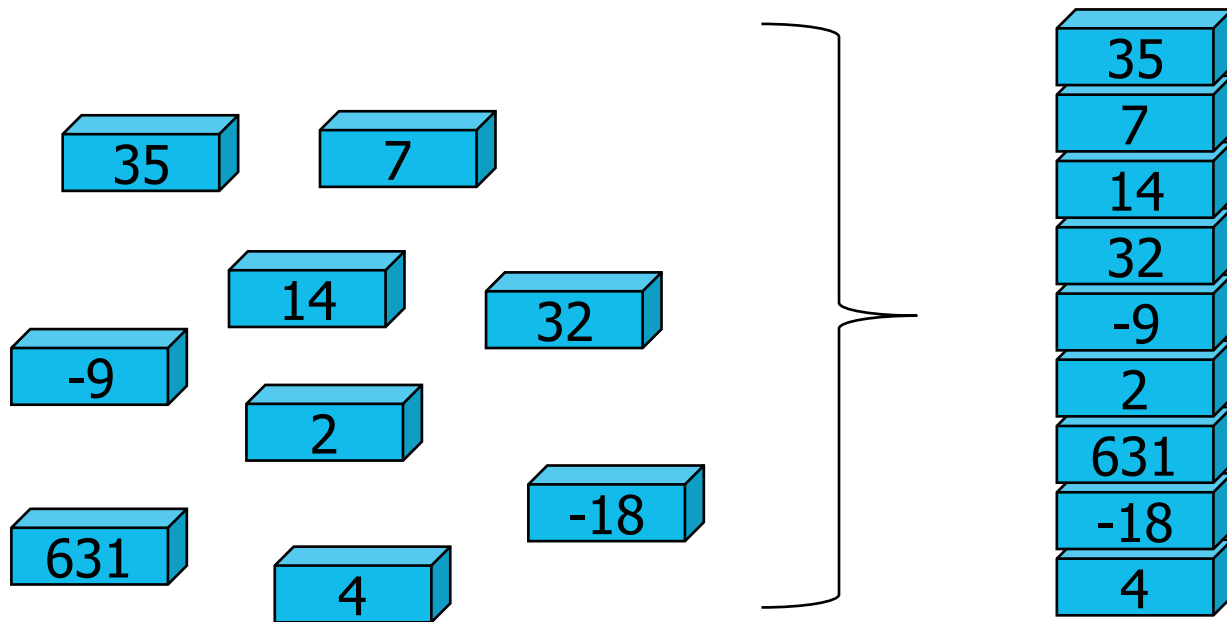
A COSA SERVONO I VETTORI: INSIEMI NON ORDINATI.
SEQUENZE ORDINATE, CORRISPONDENZA INDICE-DATO

I vettori nel problem solving

- I vettori sono insieme di dati dello stesso tipo
- Un vettore può essere utilizzato come contenitore di dati (omogenei)
 - senza alcun criterio di ordine
 - Con un criterio di ordine
 - Sfruttando la corrispondenza dato-indice

Vettore come contenitore (non ordinato)

- Il vettore è un semplice contenitore di dati.
- L'ordine non interessa



Il vettore come contenitore

■ Utilizzo:

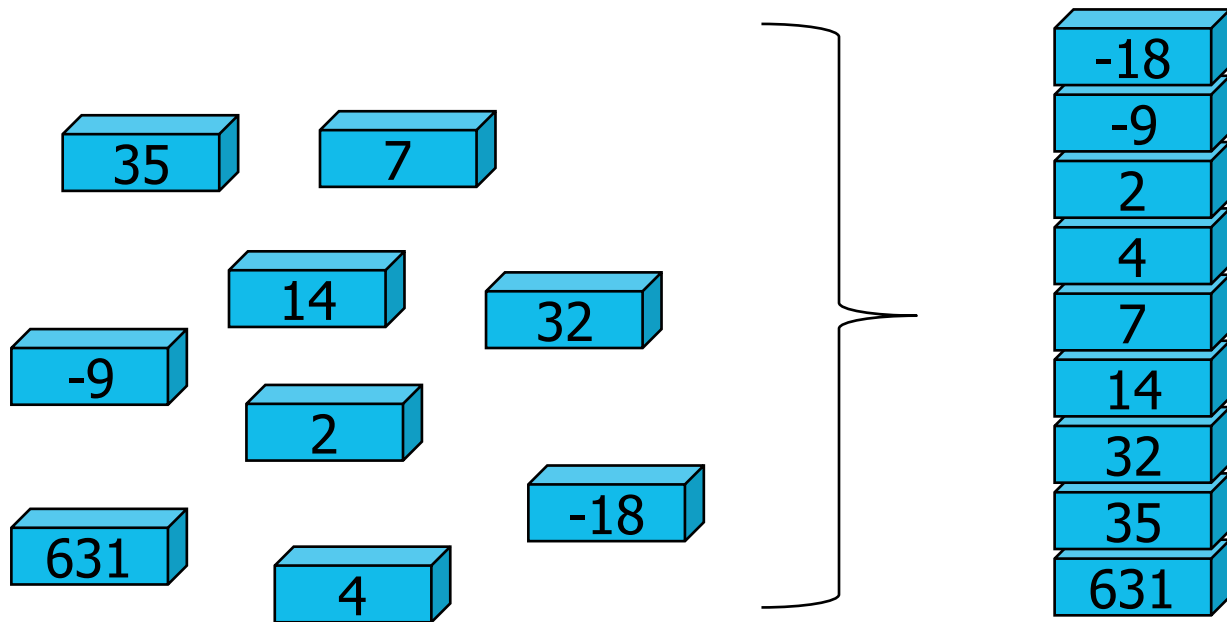
- problemi in cui è sufficiente raccogliere un insieme di dati, senza relazioni di ordine
- un dato può essere in qualunque posizione nel vettore
- accesso ai dati mediante generazione (iterativa) di tutti gli indici

Esempi:

- collezionare dati in input, per elaborazioni successive, o preparare dati per l'output
- operazioni su insiemi di dati

Vettore ordinato

- I dati nel vettore sono organizzati secondo un criterio di ordine (es. valori crescenti)



Vettore ordinato

Utilizzo:

- problemi in cui occorre ordinare e/o mantenere ordinato un insieme di dati, secondo un certo criterio
- il caso più frequente è il vettore mono-dimensionale: ordine lineare (totale o parziale)
- l'indice di un dato indica la posizione nell'ordinamento

Esempi:

- ordinare dati mediante criterio cronologico (inverso all'input) o secondo valori (crescenti/decrescenti)
- sequenze di campioni (numeri) di grandezze fisiche
- calcoli matematici e/o statistici su successioni di numeri

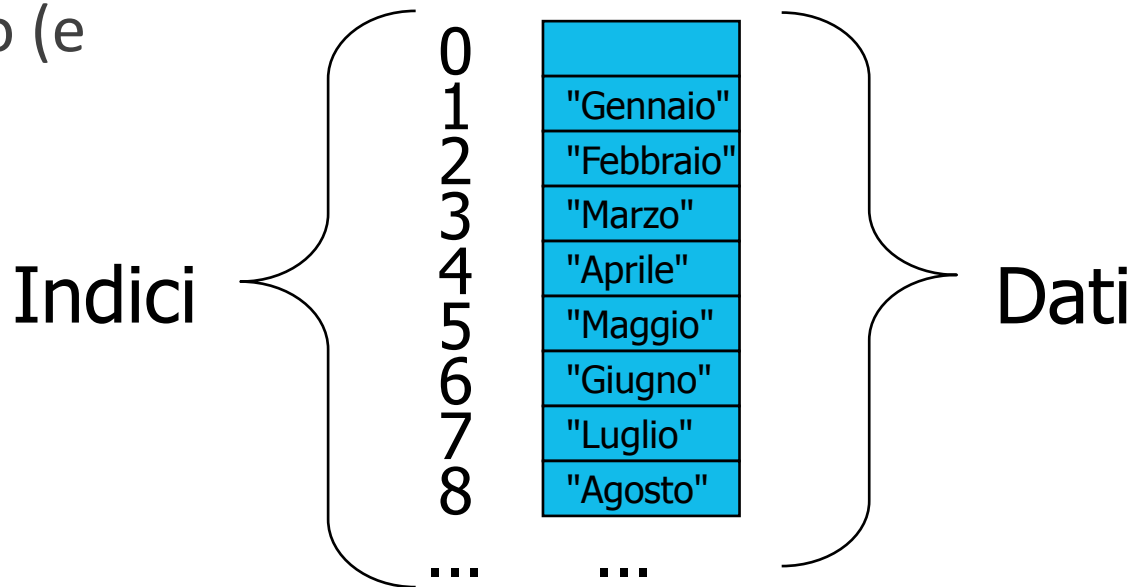
Corrispondenza indice \leftrightarrow dato

Ad ogni indice (intero
nell'intervallo $0..NDATI-1$)
corrisponde un dato (e
viceversa):

0	
1	"Gennaio"
2	"Febbraio"
3	"Marzo"
4	"Aprile"
5	"Maggio"
6	"Giugno"
7	"Luglio"
8	"Agosto"
...	...

Corrispondenza indice \leftrightarrow dato

Ad ogni indice (intero
nell'intervallo $0..NDATI-1$)
corrisponde un dato (e
viceversa):



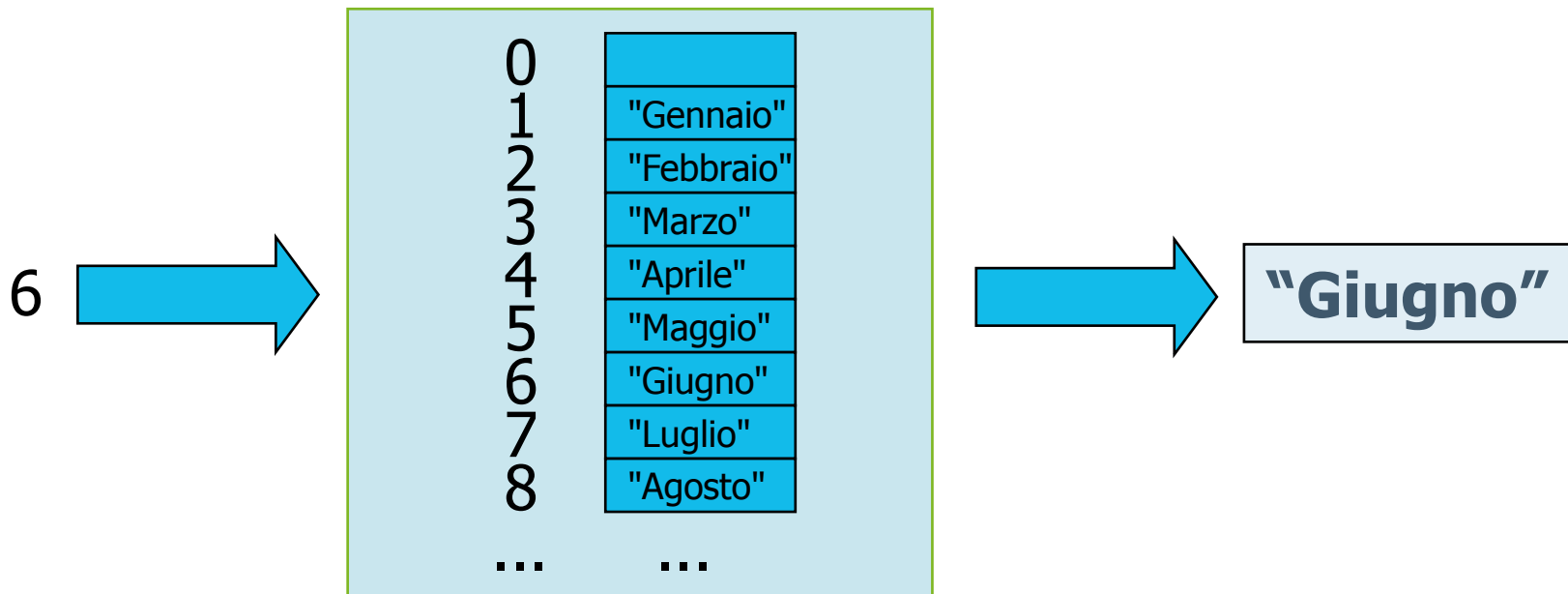
Indici e dati

La relazione indice \leftrightarrow dato:

- ogni casella di un vettore è caratterizzata da un indice (o più indici, nel caso multi-dimensionale) e da un dato
- la casella del vettore può essere quindi utilizzata per mettere in relazione indice e dato

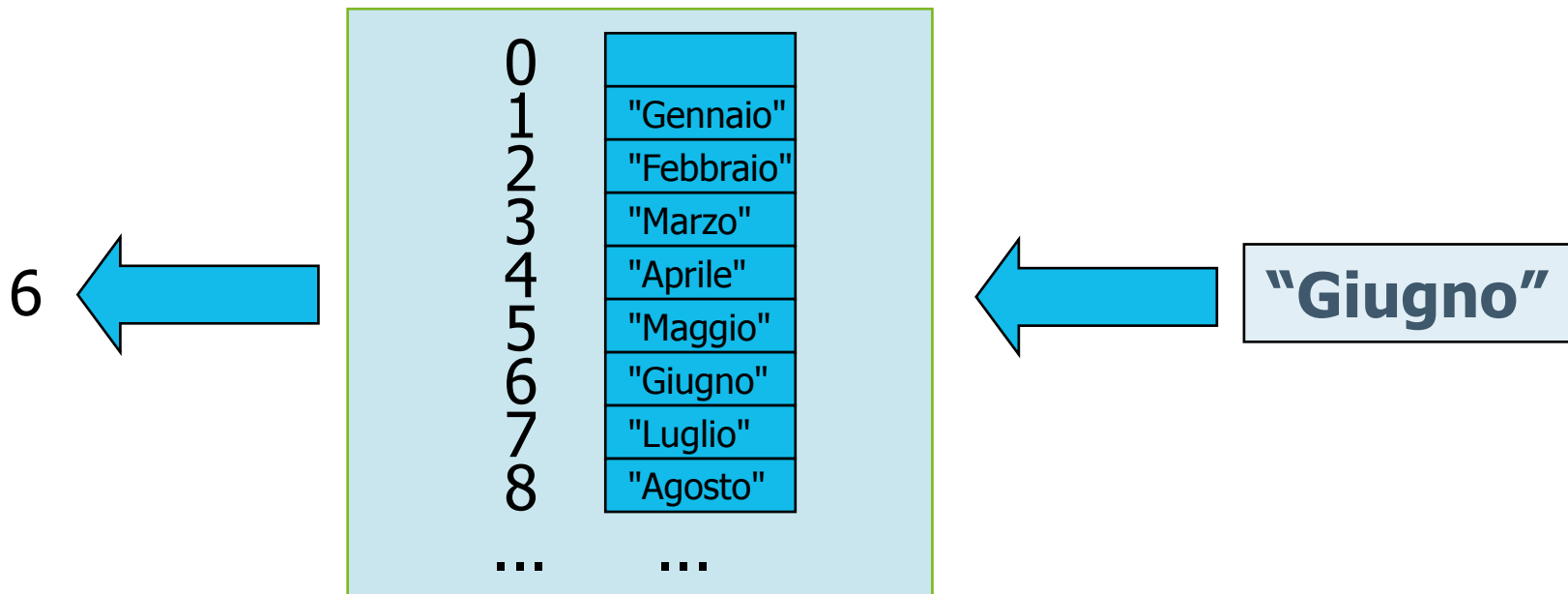
Da indice a dato

A partire dall'indice, calcolare il dato. Operazione immediata, grazie al meccanismo di accesso ai vettori:



Da dato a indice

A partire dal dato, calcolarne l'indice. Operazione non immediata, in quanto occorre cercare il dato:



■ Utilizzo:

- problemi in cui esistono relazioni/corrispondenze tra numeri interi e dati (numerici o non numerici)
- l'intero è associato all'indice di una casella, il dato al contenuto
- attenzione!
 - l'intero (indice) non può essere troppo grande
 - è possibile che occorra un dato vuoto (nullo) per le caselle non utilizzate

Esempi:

- problemi numerici (statistiche e conteggi in relazione a dati interi, tabulazione di funzione nel piano cartesiano $y=f(x)$, con ascissa intera o riconducibile a intero)
- problemi di codifica/decodifica numerica di informazioni (non numeriche)
- problemi di elaborazione testi (matrice di caratteri in corrispondenza a pagina da visualizzare)
- problemi di verifica (flag logici in corrispondenza a dati interi)
- problemi di selezione (ricerca di indice in corrispondenza a una chiave).

Problemi numerici

PROBLEMI NUMERICI CHE RICHIEDONO USO DI
VETTORI

Problemi numerici

- Problemi di algebra, geometria, statistica, ecc., simili a quelli risolti mediante dati scalari
- I vettori possono essere utilizzati:
 - per collezionare e manipolare (insiemi di) numeri
 - per rappresentare dati con struttura lineare (vettori) o tabellare (matrici)
 - per gestire corrispondenze tra numeri (indice e dato).

Problemi su insiemi di numeri

- Problemi nei quali si gestiscono insiemi (gruppi) di dati numerici, con operazioni di I/O, unione, intersezione, ... NON CONTA L'ORDINE
- La soluzione spesso si basa su costrutti iterativi (eventualmente annidati) tali da:
 - percorrere gli elementi di un insieme
 - percorrere, per ogni elemento di un insieme, tutti gli elementi di un altro insieme.

Intersezione tra insiemi di numeri

Formulazione:

- acquisire da tastiera un primo gruppo di 10 interi (privo di numeri ripetuti)
- acquisire da tastiera un secondo gruppo di 10 interi (privo di numeri ripetuti)
- calcolare e visualizzare tutti i numeri che sono presenti in entrambi i gruppi

Soluzione:

- si tratta di un problema di intersezione tra insiemi, che richiede di determinare, per ogni elemento del primo insieme, se appartiene anche al secondo

Algoritmo:

- input: due iterazioni per acquisire i dati dei due gruppi
- elaborazioni: doppia iterazione (annidata) per confrontare tutti i dati del primo gruppo con tutti quelli del secondo (o viceversa)
- output: iterazione sui dati dell'insieme intersezione.

Le tre parti possono essere distinte, oppure (parzialmente) integrate: mentre si esegue l'input si calcola l'intersezione e si scrivono i risultati in output.

Struttura dati:

La scelta dipende dall'algoritmo adottato:

- è necessario almeno un vettore per i dati del primo insieme
- l'utilizzo di un secondo vettore dipende dal fatto di adottare uno schema
 - tre fasi separate: input, elaborazioni, output
 - elaborazioni ed output fatte durante l'input del secondo gruppo (per ogni numero del secondo gruppo, si determina direttamente se appartiene anche al primo e si fornisce il relativo output)
- la soluzione proposta utilizza due vettori, con tre fasi separate. Il risultato (intersezione) viene sovrapposto al primo vettore.

Codice

```
#define NDATI 10
#include <stdio.h>

void leggivettore (int dati[], int n);
void scrivivettore (int dati[], int n);

int main(void)  i{
    int dati0[NDATI], dati1[NDATI];
    int i, j, ni, trovato;
    /* input */
    leggivettore(dati0,NDATI);
    leggivettore(dati1,NDATI);
    /* calcolo intersezione */
    for (i=ni=0; i<NDATI; i++) {
        trovato=0;
```

```
        for (j=0; j<NDATI&&(!trovato); j++) {
            if (dati0[i]==dati1[j])
                trovato=1;
        }
        /* se dato appartiene ad intersezione
           riscrivi nella parte iniziale del
           vettore dati0 (già confrontato) */
        if (trovato)
            dati0[ni++]=dati0[i];
    }
    /* output */
    scrivivettore(dati0,ni);
}
```

Codice

```
void leggiVettore (int dati[], int n) {
    int i;
    printf("Scrivi %d numeri interi (separati da spazio o a-capo):\n", n);
    for (i=0; i<n; i++) {
        scanf("%d", &dati[i]);
    }
}

void scriviVettore (int dati[], int n);
int i;
printf("I numeri sono:\n", n);
for (i=0; i<n; i++) {
    printf("%d ", dati[i]);
}
printf("\n");
}
```

Problemi su sequenze di numeri

- I problemi interessano sequenze di dati (**ordinati**) che debbono essere immagazzinati in un vettore prima di venir elaborati, perchè:
 - è necessario attendere l'ultimo dato prima di poter elaborare i dati
 - oppure sono necessarie elaborazioni (ripetute) su tutti i dati
- Non si possono trattare sequenze infinite, ma insiemi (ordinati) finiti di dati, organizzati in vettori mono- o multi-dimensionali (matrici)

Normalizzazione di dati

Formulazione: scrivere una funzione C che:

- acquisisca da file testo una sequenza di dati reali separati da spazi o da a-capo:
 - il numero N di dati non è noto a priori ma può essere sovradimensionato (valore massimo 1000)
- determini, per l' i -esimo dato d_i ($0 \leq i < N$), le medie dei dati precedenti (p_i) e dei successivi (s_i)
- scriva su un secondo file i dati, normalizzati secondo la seguente regola:
 - ogni dato (d_i) viene sostituito dalla media aritmetica tra d_i , p_i e s_i
- i nomi dei due file siano ricevuti come parametri.

Soluzione:

- occorre calcolare, in modo iterativo, le medie, quindi, con una successiva iterazione, le normalizzazioni dei dati

Struttura dati:

- un vettore per acquisire i dati dal file
- un vettore per calcolare le medie (? ... da decidere)

NOTA: Si potrebbero evitare i vettori rileggendo più volte il file

- soluzione suggerita come esercizio, ma sconsigliata: in genere meglio evitare di leggere un file molte volte.

Algoritmo 1: algoritmo $O(N^2)$

- input: vettore, riempito mediante lettura iterativa
- Elaborazioni: per ogni dato
 - calcola la media dei predecessori e dei successori (funzione avg)
 - fa la loro media (in un altro vettore (quello originale deve essere mantenuto per calcolare correttamente le medie)
 - Il predecessore del primo dato e il successore dell'ultimo mancano: Ci potrebbero essere due strategie: a) estrapolare un valore, ad es. 0 (la strategia usata) oppure un valore uguale al primo/ultimo; b) non considerare il dato e quindi fare solo la media tra 2 dati.
- output del vettore ricalcolato.

Esempio:

si supponga di ricevere i seguenti 6 valori:

- 4.0 5.0 3.0 2.0 1.0 7.0

Il risultato sarà:

- $\text{dati}[0] = (4.0 + 0.0 + (18.0)/5)/3 = 2.53$
- $\text{dati}[1] = (5.0 + 4.0/1 + (13.0)/4)/3 = 4.08$
- $\text{dati}[2] = (3.0 + 9.0/2 + (10.0)/3)/3 = 3.61$
- $\text{dati}[3] = (2.0 + 12.0/3 + (8.0)/2)/3 = 3.33$
- $\text{dati}[4] = (1.0 + 14.0/4 + (7.0)/1)/3 = 3.83$
- $\text{dati}[5] = (7.0 + 15.0/5)/3 = 3.33$

Codice

```
#define NMAX 100
/* version 1: O(N^2) */
float avg(int d[], int i0, int i1);
void normalizeNum(char nfin[],char nfout[]){
    float data[NMAX], dataNew[NMAX];
    int i, j, N;
    N = readFile(nfin,dati,NMAX); /* input */
    for (i=0; i<N; i++) {
        float pred = avg(data,0,i-1);
        float succ = avg(data,i+1,N-1);
        dataNew[i] = (pred+data[i]+succ)/3;
    }
    writeFile(nfout,datiNew,N); /* output */
}
```

```
float avg(int d[], int i0, int i1) {
    int i;
    float sum;
    if (i0>i1) {
        return 0.0; // no data: assume 0
    }
    sum = 0.0;
    for (i=i0; i<=i1; i++) {
        sum = sum + d[i];
    }
    return sum/(i1-i0+1);
}
```

Codice

```
int readFile(char fileName[], float d[], int nmax)
{
    int i;
    FILE *fp;
    fp = fopen(fileName,"r");
    if (fp==NULL)
        return 0;
    for (i=0; i<nmax; i++) {
        if (fscanf(fp, "%f", &d[i])==EOF)
            break;
    }
    return i;
}
```

```
void writeFile(char fileName[], float d[], int n)
{
    int i;
    FILE *fp;
    fp = fopen(fileName,"w");
    if (fp==NULL)
        return;
    for (i=0; i<n; i++) {
        printf("%f\n", dati[i]);
    }
}
```

Algoritmo 2: $O(N)$

- input: vettore , riempito mediante lettura iterativa
- elaborazioni: è sufficiente calcolare
 - la sommatoria di tutti dati (STOT)
 - per ogni dato, la somma di se stesso e dei predecessori (sum_i). A partire da questa è possibile calcolare:
$$p_i = \text{sum}_{i-1}/i,$$
$$s_i = (\text{STOT} - \text{sum}_i)/(N-i-1)$$
 - i valori normalizzati (sostituendo i dati sul vettore iniziale):
- output del vettore ricalcolato.

Esempio:

si supponga di ricevere i seguenti 6 valori:

- 4.0 5.0 3.0 2.0 1.0 7.0

Il vettore somme conterrà:

- 4.0 9.0 12.0 14.0 15.0 22.0

STOT = 22.0

Il risultato sarà:

- $\text{dati}[0] = (4.0 + 0.0 + (22.0-4.0)/5)/3 = 2.53$
- $\text{dati}[1] = (5.0 + 4.0/1 + (22.0-9.0)/4)/3 = 4.08$
- $\text{dati}[2] = (3.0 + 9.0/2 + (22.0-12.0)/3)/3 = 3.61$
- $\text{dati}[3] = (2.0 + 12.0/3 + (22.0-14.0)/2)/3 = 3.33$
- $\text{dati}[4] = (1.0 + 14.0/4 + (22.0-15.0)/1)/3 = 3.83$
- $\text{dati}[5] = (7.0 + 15.0/5 + 0.0)/3 = 3.33$

Codice

```
#define NMAX 1000
/* version 2: O(N) */
void normalizeNum(char nfin[],char nfout[]){
    float data[NMAX], sum[NMAX];
    int i, j, N, STOT;
    /* input */
    N = readFile(nfin,dati,NMAX); /* input */
    /* partial sums */
    sum[0]=data[0];
    for (i=1; i<N; i++)
        sum[i] = sum[i-1]+data[i];
    STOT = sum[N-1]; /* sum of all numbers */
    ...
}
```

```
/* normalize */
data[0] = (data[0] + (STOT-data[0])/(N-1))/3;
for (i=1; i<N-1; i++) {
    float pred = sum[i-1]/i;
    float succ = (STOT-sum[i])/(N-i-1);
    data[i] = (pred+data[i]+succ)/3;
}
data[N-1] = (data[N-1] + sum[N-2]/(N-1))/3;

writeFile(nfout,dati,N); /* output */
}
```


Problemi su statistiche per gruppi

- I problemi sono caratterizzati dalla suddivisione dei numeri in classi/gruppi numerabili ed identificabili da un intero
- La raccolta di conteggi o dati statistici può essere effettuata sulle caselle di un vettore
- A ogni classe o gruppo corrisponde un indice (e una casella del vettore).

Suddivisione in classi

Formulazione: scrivere una funzione C che:

- riceva come parametro un vettore di interi di valore compreso tra 0 e 100 (il secondo parametro indica la dimensione del vettore)
- raggruppi gli interi in decine, calcoli e visualizzi i conteggi dei numeri appartenenti a ciascuna decina.

Esempio:

si supponga di ricevere i seguenti 20 valori:

- 3 6 9 16 22 23 30 32 40 48 65 78 7 8 10 15 25
90 27 26

I numeri saranno raggruppati come segue:

- (3, 6, 9, 7, 8), (16, 10, 15), (22, 23, 25, 26, 27), (30, 32), (40, 48), (-),
(65), (78), (-), (90), (-)

I conteggi visualizzati saranno:

- 5, 3, 5, 2, 2, 0, 1, 1, 0, 1, 0

■ Soluzione:

- Si potrebbe fare una doppia iterazione (per ogni decina, iterare su tutti i numeri e contare quelli appartenenti alla decina): il costo sarebbe $O(\text{decine} * \text{numeri})$
- Più efficiente ($O(\text{numeri})$): utilizzare un vettore di contatori, sfruttando la corrispondenza indice-dato. Per ogni dato in ingresso si calcola l'indice corrispondente alla decina di appartenenza, e si aggiorna il relativo contatore

■ Struttura dati:

- un vettore di interi come parametro alla funzione
- un vettore di contatori (con indici da 0 a 10).

Algoritmo:

- azzeramento dei contatori, per ogni decina (le decine sono numerate da 0 a 10)
- iterazione sui dati interi. Per ognuno:
 - si calcola la decina di appartenenza con una divisione intera $d = \text{dati}[i]/10$;
 - si accede al vettore dei conteggi (utilizzando la decina come indice) incrementando il contenuto
- iterazione sui contatori per visualizzare i conteggi.

Codice

```
void contaPerDecine (int dati[], int n){
    int i, d, conta[11];
    for (i=0; i<=10; i++)
        conta[i]=0;
    for (i=0; i<n; i++) {
        d = dati[i]/10;
        conta[d] = conta[d]+1;
    }
    for (i=0; i<=10; i++)
        printf ("%d dati in decina %d \n",
                conta[i], i+1);
}
```

Problemi di codifica

PROBLEMI DI CODIFICA DI NUMERI E TESTI

Problemi di codifica di numeri

Nei problemi di codifica di numeri, i vettori possono essere utilizzati per:

- immagazzinare le cifre in una data codifica
- manipolare i numeri, lavorando a livello di codifica in cifre.

Codifica binaria di un intero

Formulazione: realizzare una funzione C che, ricevuto come parametro un intero ($0 \leq n \leq 2^{32} - 1$), ne determini la codifica binaria e visualizzi i bit.

Soluzione:

- costruzione iterativa dei bit, a partire dai meno significativi
- divisioni successive per 2, i resti delle divisioni sono i bit della codifica .

Struttura dati:

- un parametro formale (intero): n
- un vettore (bit) per i bit della codifica

Algoritmo:

- iterazione per generare, in bit, i bit, a partire dai meno significativi (al massimo 32 bit)
- iterazione per visualizzare i bit (dal più significativo).

Codice

```
void binarioVettore (int n) {  
    int i, bit[32];  
    i=0;  
    do {  
        bit[i++] = n%2;  
        n = n/2;  
    } while (n>0);  
    i--;  
    while (i>=0) {  
        printf("%d",bit[i--]);  
    }  
    printf("\n");  
}
```

Codice

```
void binarioVettore (int n) {  
    int i, bit[32];  
    i=0;  
    do {  
        bit[i++] = n%2;  
        n = n/2;  
    } while (n>0);  
    i--;  
    while (i>=0) {  
        printf("%d",bit[i--]);  
    }  
    printf("\n");  
}
```

Calcola bit meno significativo

Codice

```
void binarioVettore (int n) {  
    int i, bit[32];  
    i=0;  
    do {  
        bit[i++] = n%2;  
        n = n/2;  
    } while (n>0);  
    i--;  
    while (i>=0) {  
        printf("%d",bit[i--]);  
    }  
    printf("\n");  
}
```

Continua finchè n non è 0
Se inizialmente n=0 calcola 1 bit

Problemi di codifica/decodifica di testi

Problemi di codifica, ricodifica o crittografia applicati a testi, nei quali un vettore può essere utilizzato come:

- tabella di codifica o ricodifica
 - mediante la corrispondenza indice-dato
 - come insieme di coppie dato-codice o codice0-codice1
- contenitore (ordinato o non) per dati intermedi.

Crittografia di un file di testo

Formulazione:

- crittografare il contenuto di un file testo, immagazzinando il risultato in un file risultato
- i codici dei caratteri vengono modificati, in base alla tabella di ricodifica contenuta in un file:
 - ogni riga del file contiene due numeri interi (compresi tra 0 e 255), che rappresentano, rispettivamente il codice ASCII di un carattere e il codice ASCII del carattere ricodificato
 - i codici non presenti non vanno modificati
 - la tabella garantisce l'univocità della ricodifica.

Soluzione:

- leggere la tabella da file, costruendo una vettore di ricodifica indice (codice iniziale) → dato (nuovo codice)
- leggere iterativamente i caratteri dal primo file
 - calcolare la ricodifica del carattere
 - scrivere il carattere sul file risultato

Struttura dati:

- tre variabili di tipo puntatore a FILE per gestire i due file in lettura e quello in scrittura; una stringa per i nomi dei file
- un vettore di caratteri per la tabella di ricodifica
- una variabile char per lettura e ricodifica dei caratteri

Algoritmo:

- acquisizione dei nomi di file e loro apertura
- inizializzazione della tabella di codifica (per codici invariati)
- lettura tabella di ricodifica
- iterazione di lettura di un carattere, ricodifica, scrittura nel secondo file
 - la ricodifica viene fatta mediante passaggio indice → dato nella tabella

Codice

```
#define MAXRIGA 30
int main(void) {
    char ch, nomefile[MAXRIGA];
    char tabella[256];
    FILE *fpin, *fpout, *ftab;
    int i, nuovo;
    printf("nome file in ingresso: ");
    scanf("%s", nomefile);
    fpin = fopen(nomefile, "r");
    printf("nome file in uscita: ");
    scanf("%s", nomefile);
    fpout = fopen(nomefile, "w");
    printf("nome file tabella: ");
    scanf("%s", nomefile);
    ftab = fopen(nomefile, "r");
```

```
/* inizializza tabella: ogni carattere
   convertito in se stesso */
for (i=0; i<256; i++)
    tabella[i] = (char)i;
/* leggi da file le conversioni presenti */
while (fscanf(ftab, "%d%d", &i, &nuovo) == 2)
    tabella[i] = (char)nuovo;
/* converti fpin in fpout */
while (fscanf(fpin, "%c", &ch) == 1) {
    fprintf(fpout, "%c", tabella[(int)ch]);
}
fclose(fpin); fclose(fpout); fclose(ftab);
}
```

Problemi di text-processing

PROBLEMI ELABORAZIONE TESTI: INPUT,
MODIFICA, OUTPUT

Problemi di text-processing

- Problemi nei quali occorre manipolare sequenze di caratteri e/o stringhe

Esempi: input (e comprensione) di un testo, costruzione o modifica di testo, creazione di messaggio in un dato formato

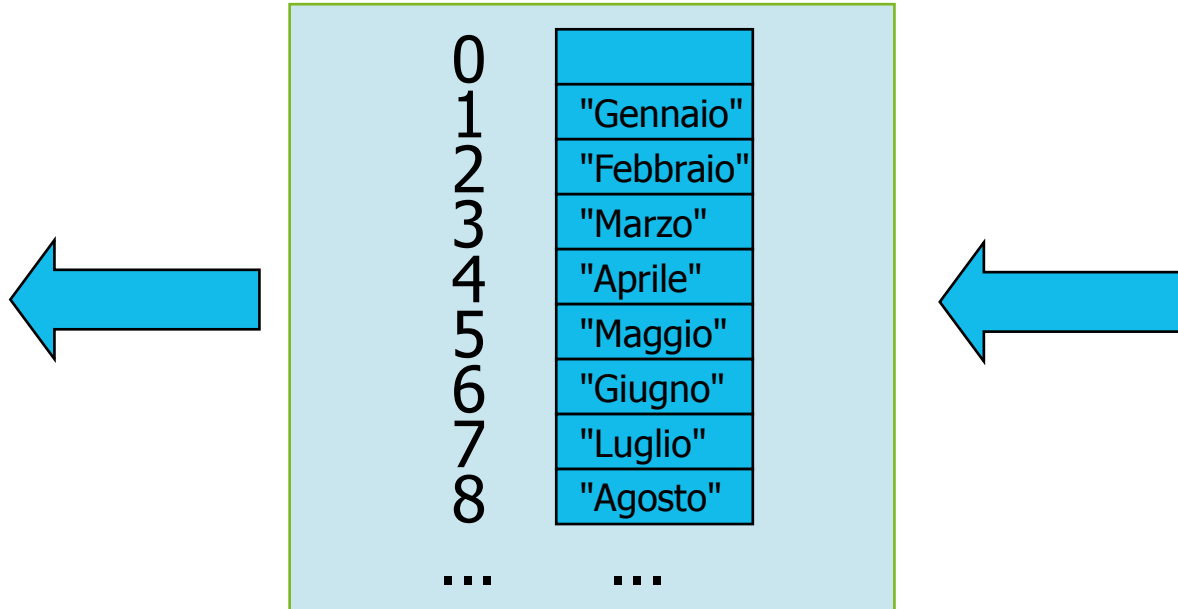
- Vettori (o matrici) di caratteri (o di stringhe) sono spesso necessari per:
 - generazione di testi a partire da regole o funzioni
 - trasformazione di testi esistenti.
 - acquisire in input o predisporre per output un testo

Vettori di stringhe e selezione

- Un vettore di stringhe è formalmente una matrice di caratteri
 - `char parole[NPAROLE][MAXL];`
- Problema: data una stringa (una parola), cercarla in una tabella (di parole) per "capire" a quale dato (es. un numero) corrisponda
- La selezione basata direttamente su stringhe richiede confronti (`strcmp`) e costrutti condizionali `if` (non sono possibili `switch`)
- I vettori (usati come tabelle) possono consentire la traduzione da codici testuali a codici numerici, con cui:
 - è possibile la selezione mediante `switch`
 - si ottiene una migliore gestione/organizzazione dei casi da trattare
 - si ottengono informazioni più compatte, trasferibili tra moduli, come parametri o valori di ritorno di funzioni (es. codici di errore).

La tabella di conversione (A)

Per la conversione da stringa a intero si può utilizzare la corrispondenza dato \rightarrow indice

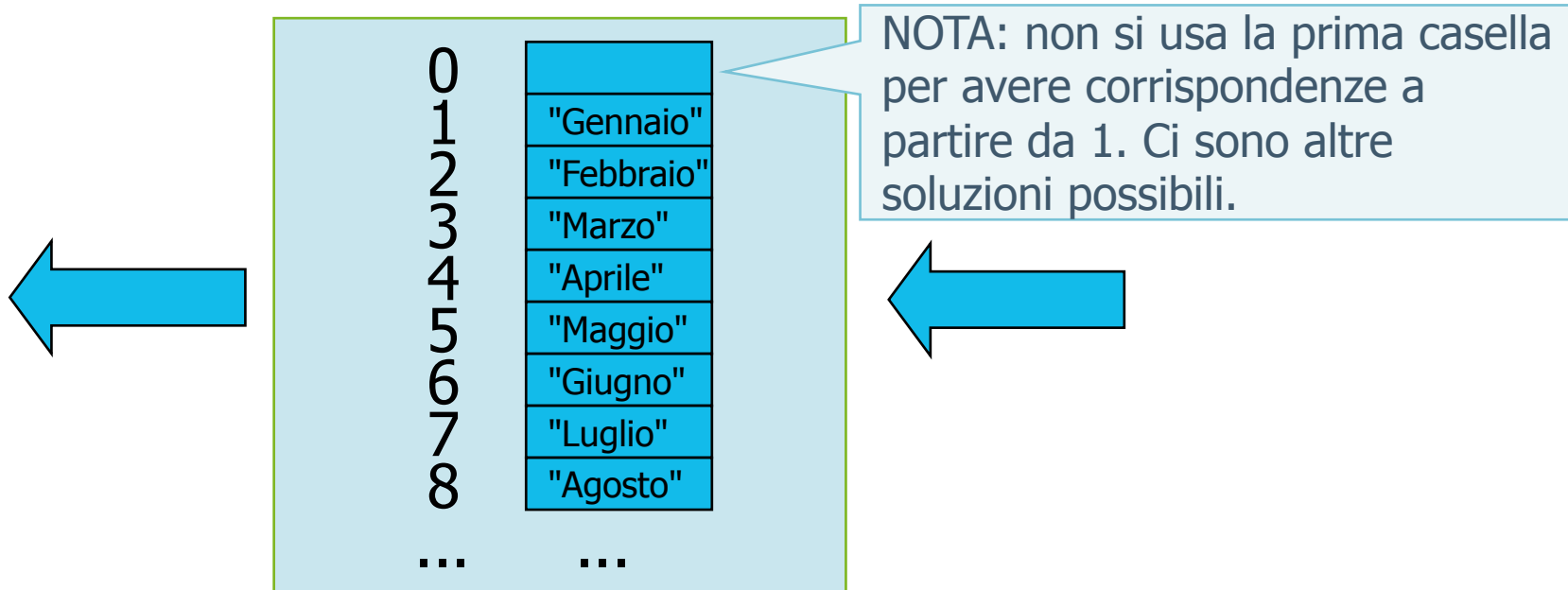


The diagram shows a light blue rectangular box containing a table. To the left of the box is a large blue arrow pointing left. To the right of the box is another large blue arrow pointing left. The table inside the box has two columns: the first column contains indices from 0 to 8, followed by an ellipsis (...); the second column contains month names: an empty cell, "Gennaio", "Febbraio", "Marzo", "Aprile", "Maggio", "Giugno", "Luglio", "Agosto", followed by an ellipsis (...).

0	
1	"Gennaio"
2	"Febbraio"
3	"Marzo"
4	"Aprile"
5	"Maggio"
6	"Giugno"
7	"Luglio"
8	"Agosto"
...	...

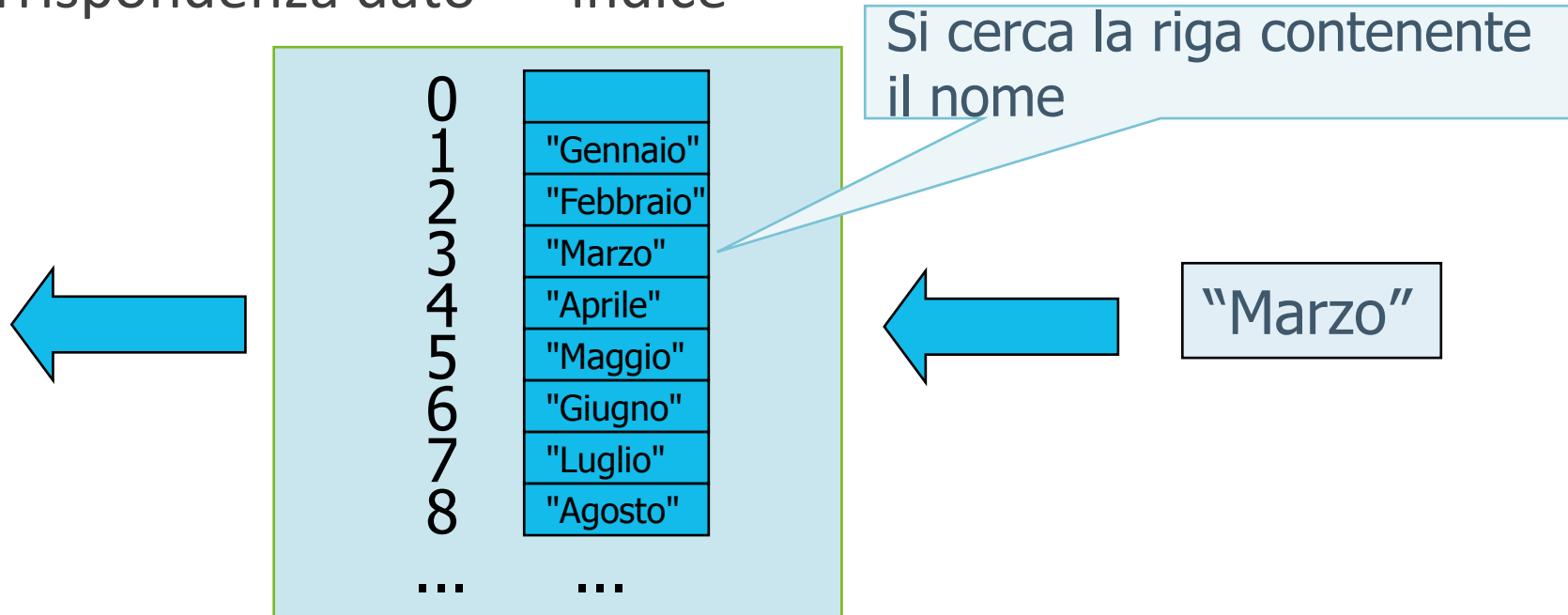
La tabella di conversione (A)

Per la conversione da stringa a intero si può utilizzare la corrispondenza dato \rightarrow indice



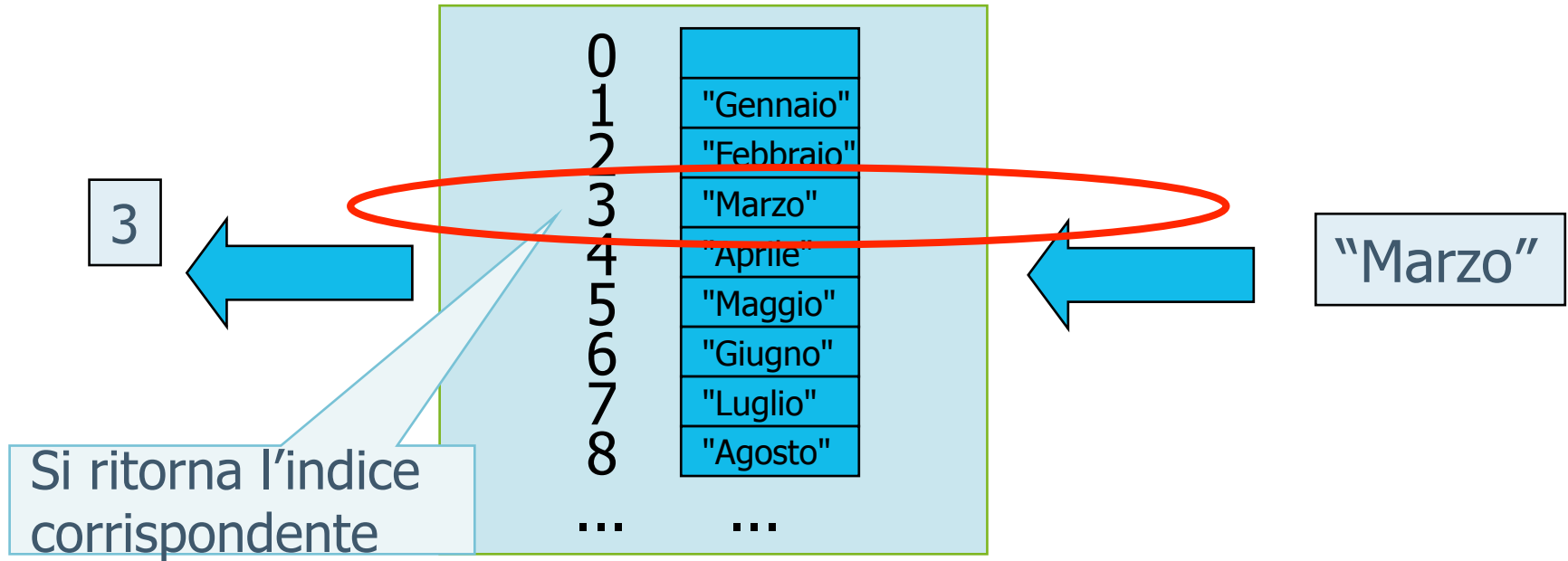
La tabella di conversione (A)

Per la conversione da stringa a intero si può utilizzare la corrispondenza dato \rightarrow indice



La tabella di conversione (A)

Per la conversione da stringa a intero si può utilizzare la corrispondenza dato \rightarrow indice



```
#include <string.h>

int monthStringToNum (char month[]) {
    char table[13][10] = {"",
                          "January", "February", "March", "April", "May", "June",
                          "July", "August", "September", "October", "November", "December"};

    int i;
    for (i=1; i<=12; i++) {
        if (strcmp(month, table[i])==0) {
            return i; // found: return index
        }
    }
    return -1; // there is a problem, month not found
}
```

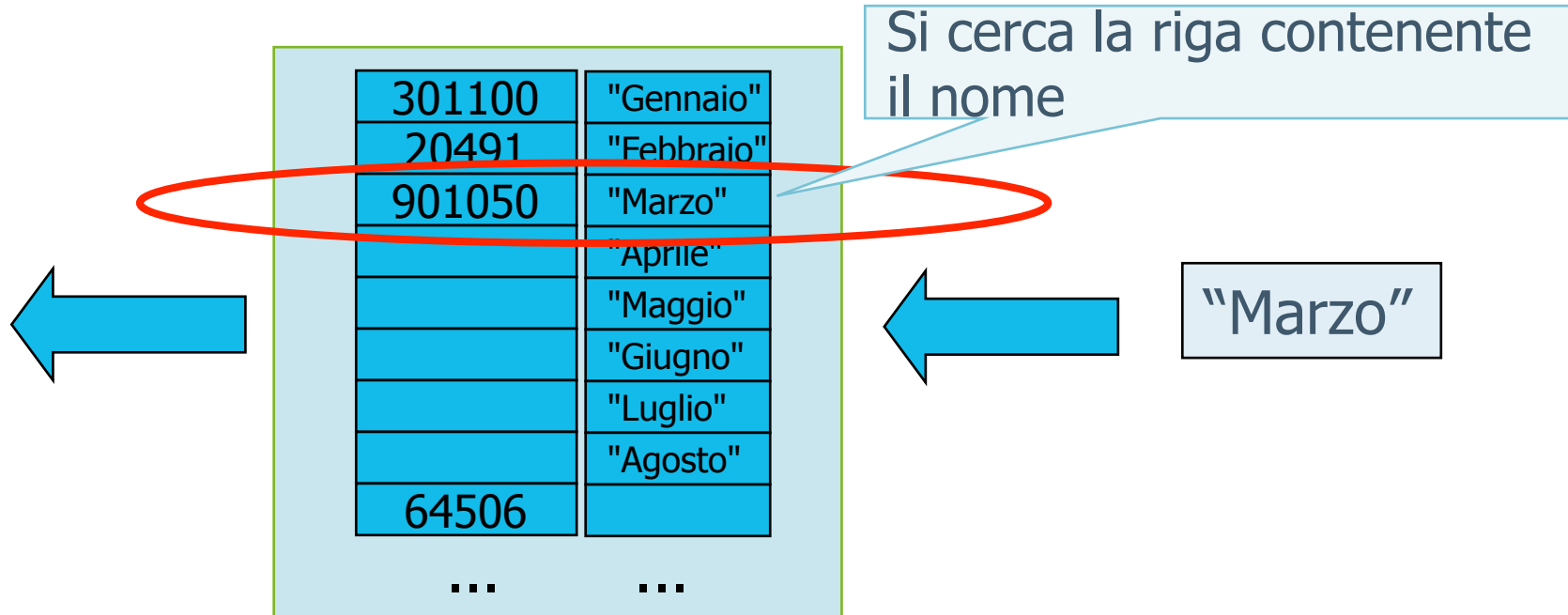
La tabella di conversione (B)

Se i valori interi sono troppo grandi (non adatti come indici) si può realizzare un vettore di `struct` (codice,nome) o un doppio vettore



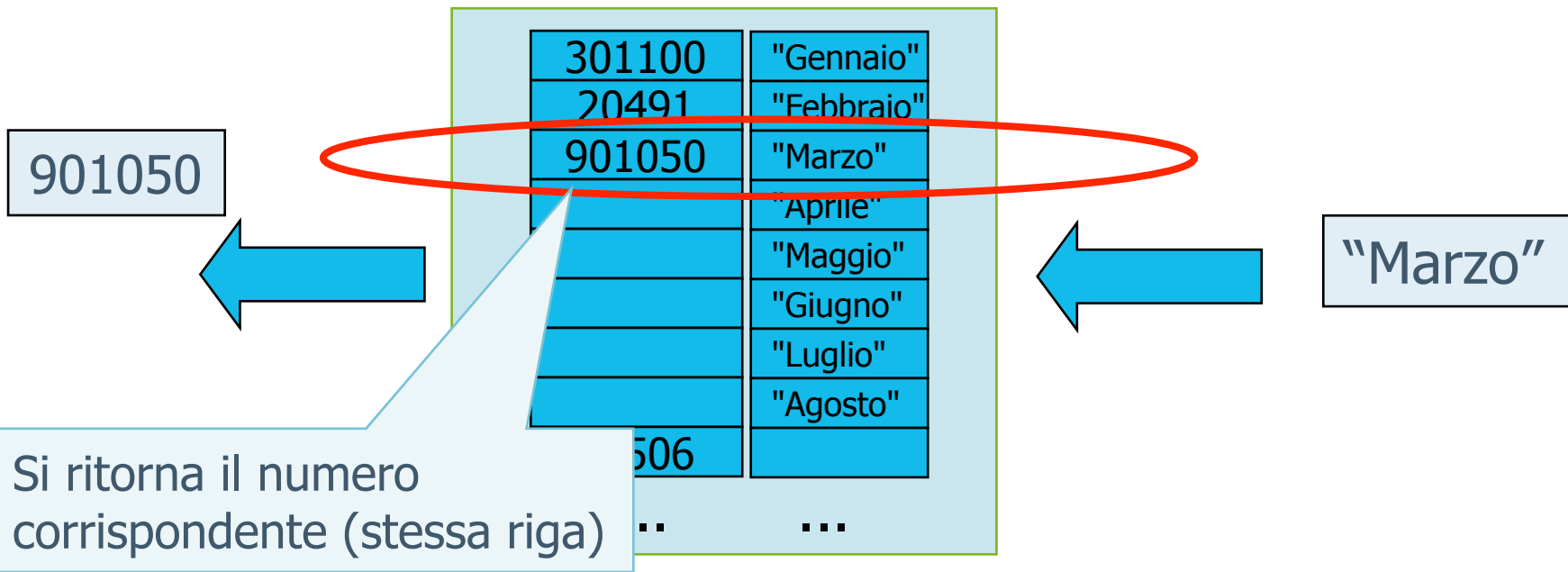
La tabella di conversione (B)

Se i valori interi sono troppo grandi (non adatti come indici) si può realizzare un vettore di `struct` (codice,nome) o un doppio vettore



La tabella di conversione (B)

Se i valori interi sono troppo grandi (non adatti come indici) si può realizzare un vettore di `struct` (codice,nome) o un doppio vettore



```

struct monthEntry {
    int num;
    char name[10];
}

int main (void) {
    int i, num;
    char month[10];
    struct monthEntry table[12];
    if (readTable(table) != 0)
        do {
            printf("write a month"); scanf("%s",month);
            n = monthStringToNum(table,month);
            if (n>=0)
                printf("month: %s -> num: %d\n", month, n);
        } while (n>=0);
    return 0;
}

```

```

int readTable (struct monthEntry t[12]) {
    FILE *fp; int i;
    fp = fopen("table.txt","r");
    if (fp==NULL) {
        printf("Error opening table.txt\n"); return 0;
    }
    for (i=0; i<12; i++)
        fscanf(fp,"%d%s", &t[i].num, t[i].name);
    fclose(fp); return 1;
}

int monthStringToNum (struct monthEntry t[12], char m[]) {
    int i;
    for (i=0; i<12; i++) {
        if (strcmp(month,t[i].name)==0)
            return t[i].num; // found: return num
    }
    return -1; // there is a problem, month not found
}

```

Menu con scelta su una parola

- Formulazione: scrivere una funzione che, iterativamente, acquisisca da tastiera una stringa (al massimo 50 caratteri, contenente eventuali spazi):
 - la prima parola diversa da spazio costituisce il selettore
 - se la parola è “fine”, occorre terminare l’iterazione
 - se la parola è una tra “cerca”, “modifica”, “stampa” (ignorare differenza maiuscole/minuscole), occorre attivare, rispettivamente, le funzioni cerca, sostituisci, stampa, passando loro come parametro il resto della stringa (oltre la parola di selezione)
 - ogni altra parola va segnalata come errata.
- Soluzione: corrispondenza dato-indice

- Modularizzazione:
 - funzioni di input e conversione da stringa a codice
- Tabella:
 - vettore inizializzato mediante (puntatori a) costanti stringa
- Conversione da stringa a codice:
 - iterazione di ricerca su vettore

Codice

```
#include <stdio.h>
#include <string.h>

#define c_cerca 0
#define c_modifica 1
#define c_stampa 2
#define c_fine 3
#define c_err 4
const int MAXL=51;

int leggiComando (void);
void menuParola (void);

void cerca (char r[]) { printf("cerca: %s\n", r); }
void modifica (char r[]) { printf("modifica: %s\n", r); }
void stampa (char r[]) { printf("stampa: %s\n", r); }
```

```
void strToLower(char s[]) {
    int i, l = strlen(s);
    for (i=0; i<l; i++)
        s[i]=tolower(s[i]);
}

int main(void) {
    menuParola();
}
```

Codice

```
#include <stdio.h>
#include <string.h>

#define c_cerca 0
#define c_modifica 1
#define c_stampa 2
#define c_fine 3
#define c_err 4
const int MAXL=51;

int leggiComando (void);
void menuParola (void);

void cerca (char r[]) { printf("cerca: %s\n", r); }
void modifica (char r[]) { printf("modifica: %s\n", r); }
void stampa (char r[]) { printf("stampa: %s\n", r); }
```

```
void strToLower(char s[]) {
    int i, l = strlen(s);
    for (i=0; i<l; i++)
        s[i]=tolower(s[i]);
}
```

```
int main(void) {
    menuParola();
}
```

Definizione di costanti più intuitive e facili da usare rispetto a corrispondenti agli interi da 0 a 4

Codice

```
void menuParola (void){
    int comando;
    char riga[MAXL];
    int i, continua=1;
    while (continua) {
        comando = leggiComando();
        fgets(riga,MAXL,stdin); /* resto della riga */
        switch (comando) {
            case c_cerca: cerca(riga); break;
            case c_modifica: modifica(riga); break;
            case c_stampa: stampa(riga); break;
            case c_fine: continua=0; break;
            case c_err:
            default: printf("comando errato\n");
        }
    }
}
```

```
int leggiComando (void) {
    int c;
    char cmd[MAXL];
    char tabella[4][9] = {
        "cerca","modifica","stampa","uscita"
    };
    printf("comando (cerca/modifica);");
    printf("/stampa/uscita): ");
    scanf("%s",cmd); strToLower(cmd);
    c=c_cerca;
    while(c<c_err && strcmp(cmd,tabella[c])!=0)
        c++;
    return (c);
}
```

Codice

```
void menuParola (void){
    int comando;
    char riga[MAXL];
    int i, continua=1;
    while (continua) {
        comando = leggiComando();
        fgets(riga,MAXL,stdin); /* resto della riga */
        switch (comando) {
            case c_cerca: cerca(riga); break;
            case c_modifica: modifica(riga); break;
            case c_stampa: stampa(riga); break;
            case c_fine: continua=0; break;
            case c_err:
            default: printf("comando errato\n");
        }
    }
}
```

Lettura comando e conversione da nome a numero

```
char tabella[4][9] = {
    "cerca","modifica","stampa","uscita"
};
printf("comando (cerca/modifica");
printf("/stampa/uscita): ");
scanf("%s",cmd); strToLower(cmd);
c=c_cerca;
while(c<c_err && strcmp(cmd,tabella[c])!=0)
    c++;
return (c);
}
```

Codice

```
void menuParola (void){
    int comando;
    char riga[MAXL];
    int i, continua=1;
    while (continua) {
        comando = leggiComando();
        fgets(riga,MAXL,stdin); /* resto della riga */
        switch (comando) {
            case c_cerca: cerca(riga); break;
            case c_modifica: modifica(riga); break;
            case c_stampa: stampa(riga); break;
            case c_fine: continua=0; break;
            case c_err:
            default: printf("comando errato\n");
        }
    }
}
```

Resto della riga, usato per eseguire il comando

```
char tabella[4][9] = {
    "cerca","modifica","stampa","uscita"
};
printf("comando (cerca/modifica");
printf("/stampa/uscita): ");
scanf("%s",cmd); strToLower(cmd);
c=c_cerca;
while(c<c_err && strcmp(cmd,tabella[c])!=0)
    c++;
return (c);
}
```

Codice

```
void menuParola (void){
    int comando;
    char riga[MAXL];
    int i, continua=1;
    while (continua) {
        comando = leggiComando();
        fgets(riga,MAXL,stdin); /* resto della riga */
        switch (comando) {
            case c_cerca: cerca(riga); break;
            case c_modifica: modifica(riga); break;
            case c_stampa: stampa(riga); break;
            case c_fine: continua=0; break;
            case c_err:
            default: printf("comando errato\n");
        }
    }
}
```

Selezione della funzione da chiamare o azione da eseguire

```
        tabella[4][9] = {
            c_cerca,"modifica","stampa","uscita"
        };
        printf("comando (cerca/modifica");
        printf("/stampa/uscita): ");
        scanf("%s",cmd); strToLower(cmd);
        c=c_cerca;
        while(c<c_err && strcmp(cmd,tabella[c])!=0)
            c++;
        return (c);
    }
```

Codice

```
void menuParola (void){
    int comando;
    char riga[MAXL];
    int i, continua=1;
    while (continua) {
        comando = leggiComando();
        fgets(riga,MAXL,stdin); /* resto della riga */
        switch (comando) {
            case c_cerca: cerca(riga); break;
            case c_modifica: modifica(riga); break;
            case c_stampa: stampa(riga); break;
            case c_fine: continua=0; break;
            case c_err:
            default: printf("comando errato\n");
        }
    }
```

Da nome a numero mediante
ricerca in tabella

```
int leggiComando (void) {
    int c;
    char cmd[MAXL];
    char tabella[4][9] = {
        "cerca","modifica","stampa","uscita"
    };
    printf("comando (cerca/modifica");
    printf("/stampa/uscita): ");
    scanf("%s",cmd); strToLower(cmd);
    c=c_cerca;
    while(c<c_err && strcmp(cmd,tabella[c])!=0)
        c++;
    return (c);
}
```

MENU: variante con tipo enum

- Il tipo enum in C:
 - Associa automaticamente nomi ai numeri interi a partire da 0
 - Es.: `enum semaforo {verde,rosso,giallo};`
Definisce il tipo "enum semaforo", che associa automaticamente i nomi verde, rosso e giallo ai numeri 0, 1, 2
 - Si possono definire altre associazioni (saltare degli interi), noi lo evitiamo
 - Attenzione, in C si può fare aritmetica, in C++ NO
 - Spesso associato a typedef, l'equivalente della #define, applicato ai tipi.
Es.: `typedef enum {verde,rosso,giallo} semaforo_e;`
- Nel menu, usare enum invece di definire costanti numeriche mediante #define

Codice

```
typedef enum {
    c_cerca,c_modifica,c_stampa,c_fine,c_err
} t_comandi;
...
void menuParola (void){
    t_comandi comando;
    char riga[MAXL];
    int i, continua=1;
    while (continua) {
        comando = leggiComando();
        fgets(riga,MAXL,stdin); /* resto della riga */
        switch (comando) {
            case c_cerca: cerca(riga); break;
            ...
        }
    }
}
```

```
t_comandi leggiComando (void) {
    t_comandi c;
    char cmd[MAXL];
    char tabella[c_err][9] = {
        "cerca","modifica","stampa","uscita"
    };
    printf("comando (cerca/modifica");
    printf("/stampa/uscita): ");
    scanf("%s",cmd); strToLower(cmd);
    c=c_cerca;
    while(c<c_err && strcmp(cmd,tabella[c])!=0)
        c++;
    return (c);
}
```

Elaborazione testi a livello carattere

- Un testo può essere costruito o modificato a livello di caratteri utilizzando un vettore o una matrice come:
 - rappresentazione (a caratteri) del testo da esaminare
 - area dati temporanea per costruire o manipolare una stringa o una matrice di caratteri.

- Elaborazione di parole o frasi come:
 - una parola (o una frase), può essere analizzata a livello di singoli caratteri
 - un vettore si rende necessario se occorre accedere direttamente ai caratteri

Esempi:

- verifica di palindromia
- taglia e incolla parte di stringa da una prima ad una seconda collocazione
- ricerca/sostituzione di sottostringa.

Controllo di palindromia

- Formulazione: si realizzi una funzione C in grado di verificare se una stringa, ricevuta come parametro, sia o meno palindroma (trascurando la differenza tra caratteri maiuscoli e minuscoli):
 - una parola è palindroma se letta dall'ultimo al primo carattere, non varia

Esempi: Anna, madam, otto, abcdefgFEDCBA
- Algoritmo:
 - Iterazione con confronto tra caratteri corrispondenti (primo-ultimo, secondo-penultimo, ...)
- Strutture dati:
 - il vettore è la stringa stessa ricevuta come parametro
 - due indici identificano i caratteri da confrontare
 - un flag implementa la quantificazione

Codice

```
int palindroma (char parola[]) {  
    int i, n, pal=1;  
  
    n = strlen(parola);  
    for (i=0; i<n/2 && pal; i++) {  
        if (toupper(parola[i]) != toupper(parola[n-1-i]))  
            pal = 0;  
    }  
    return pal;  
}
```

Costruzione di figure/grafici

Le figure o grafici rappresentati su video visto come matrice di caratteri di 25 righe e 80 colonne, possono essere preparate su una matrice di caratteri:

- ciò consente di costruire la figura senza rispettare la successione di righe che caratterizza l'output sequenziale (su video o su file testo)
- la figura viene preparata su una matrice di caratteri, sfruttando l'accesso diretto ad ogni casella
- la figura viene successivamente stampata seguendo la successione sequenziale tra righe.

Visualizzazione di parabola

■ Formulazione:

- data la parabola di equazione

$$y = ax^2 + bx + c = 0$$

- si scriva un programma che:
 - acquisisca da tastiera i coefficienti a, b, c, e i valori degli estremi (xmin, xmax) e (ymin, ymax), rispettivamente di un intervallo per le ascisse e per le ordinate
 - stampi, in un rettangolo di 20 righe per 70 colonne, un grafico (con asse delle ascisse orizzontale) che rappresenti la funzione nel rettangolo del piano cartesiano compreso negli intervalli [xmin,xmax], [ymin,ymax]

Esempio: se si acquisissero da tastiera i valori:

a=1.0, b=2.0, c=1.0, x0=-1.0, xn=4.0, ymin=-1.0, ymax=10.0 il contenuto del file sarebbe:

[illegible]

- Struttura dati: sono sufficienti variabili scalari, per rappresentare
 - coefficienti: a, b, c (float)
 - intervalli: $xmin, xmax, ymin, ymax$ (float)
 - dati intermedi: $passoX$ $passoY$ (lunghezza degli intervalli), x, y (float)
 - indici: i, j (int)

È necessaria una matrice (di char) per costruire il grafico.

■ Algoritmo:

- input dati e calcolo passo (= lunghezza intervalli)
- inizializzazione a tutti spazi della matrice di caratteri
- iterazione su valori di x
 - calcolo $y(x)$
 - se è nell'intervallo $[ymin, ymax]$ converti in intero (j) e assegna '*' nella matrice
- iterazioni su righe e colonne, per stampare matrice.

Codice

```
#include <stdio.h>
#include <math.h>
const int NR=20, NC=80;
int main(void) {
    float a, b, c, x, y, passoX, passoY,
          xmin, xmax, ymin, ymax;
    int i, j;
    char pagina[NR][NC];
    FILE *fpout = fopen("out.txt", "w");
    printf("Coefficienti (a b c): ");
    scanf("%f%f%f", &a, &b, &c);
    printf("Intervallo ascisse (xmin xmax): ");
    scanf("%f%f", &xmin, &xmax);
    printf("Intervallo ordinate (ymin ymax): ");
    scanf("%f%f", &ymin, &ymax);
```

```
/* inizializza matrice */
for (i=0; i<NR; i++)
    for (j=0; j<NC; j++)
        pagina[i][j] = ' ';
passoX = (xmax-xmin)/(NC-1);
passoY = (ymax-ymin)/(NR-1);
/* calcola punti della parabola */
for (j=0; j<NC; j++) {
    x = xmin + j*passoX;
    y = a*x*x + b*x + c;
    if (y>=ymin && y<=ymax) {
        i = (y-ymin)/passoY;
        pagina[i][j] = '*';
    }
}
```

```
/* stampa matrice per righe:  
   il minimo valore di y (prima riga) stampato per ultimo in basso */  
for (i=NR-1; i>=0; i--) {  
    for (j=0; j<NC; j++)  
        fprintf(fpout,"%c",pagina[i][j]);  
    fprintf(fpout,"\n");  
}  
fclose(fpout);  
}
```

Elaborazione testi a livello di stringhe

Un testo può essere costruito o modificato a livello di stringhe se:

- è possibile identificare sottostringhe (sequenze di caratteri) sulle quali applicare operazioni di tipo unitario
- le operazioni su stringhe debbono trovarsi in libreria, oppure essere chiamate funzioni realizzate dal programmatore.

Un vettore può essere necessario per costruire o immagazzinare temporaneamente stringhe da elaborare.

Formattazione di testo

■ Formulazione:

- è dato un file testo, le cui righe sono scomponibili in sottostringhe (di non più di 20 caratteri) separate da spazi (oppure '\t' o '\n')
- si realizzi una funzione C che, letto il file, ne copi il contenuto in un altro file, dopo aver:
 - ridotto le sequenze di più spazi ad un solo spazio
 - inserito (in sostituzione di spazi) o eliminato caratteri a-capo ('\n') in modo tale che ogni riga abbia la massima lunghezza possibile, minore o uguale a l_{max} (terzo parametro della funzione)
 - centrato il testo rispetto alla lunghezza l_{max} .

■ Soluzione:

- la soluzione è simile al problema del Cap. 3 (senza centratura del testo)
- per effettuare la centratura occorre tutta una riga di testo prima di stamparla (per calcolare il numero di spazi da stampare):
 - si può utilizzare un vettore come buffer (area temporanea)
 - la centratura (su l_{\max} caratteri) di una riga di l caratteri, si effettua stampando, prima della riga, $(l_{\max} - l) / 2$ spazi

Codice

```
#include <string.h>
const int STRLEN=21;
const int LMAX=255;
...
void format(char nin[],char nout[],int lmax){
    FILE *fin=fopen(nin,"r");
    FILE *fout=fopen(nout,"w");
    char parola[STRLEN], riga[LMAX];
    int i,l=-1; // -1 per assorbire il primo 1+strlen
    riga[0] = '\\0'; // oppure strcpy(riga,"");
    while (fscanf(fin,"%s",parola)==1) {
        if (l+1+strlen(parola) <= lmax) {
            if (l>0) { strcat(riga," "); }
            strcat(riga,parola);
            l += 1+strlen(parola);
```

```
        }
    }
    else {
        for (i=0; i<(lmax-1)/2; i++)
            fprintf(fout," ");
        fprintf(fout,"%s\\n",riga);
        strcpy(riga,parola);
        l=strlen(parola);
    }
}
}
```


Problemi di verifica e selezione

PROBLEMI DI VERIFICA, SELEZIONE E ORDINAMENTO
APPLICATI A VETTORI

Problemi di verifica e selezione

- I problemi di verifica consistono nel decidere se un insieme di informazioni o dati rispettino un dato criterio di accettazione
- Selezionare significa verificare i dati e scegliere quelli che corrispondono al criterio di verifica
- La ricerca è una delle modalità di selezione:
 - spesso si cerca il dato che corrisponde a un criterio
 - talvolta i dati possono essere molteplici.
- I vettori possono essere utilizzati:
 - come contenitori per l'insieme di dati, su cui applicare il criterio di verifica
 - come insieme dei dati tra i quali ricercare/selezionare.

Verifiche su sequenze

- Verificare una sequenza di dati significa decidere se la sequenza rispetta un criterio di accettazione
- Un vettore può essere necessario nel caso di criterio di accettazione che richieda elaborazioni su tutti i dati.

Verifica di dati ripetuti

■ Formulazione:

- un file testo contiene una sequenza di dati numerici (reali)
 - la prima riga del file indica (mediante un intero) quanti sono i dati nella sequenza
 - seguono i dati, separati da spazi o a-capo
- si scriva una funzione C che, ricevuto come parametro il puntatore al file (già aperto), verifichi che ogni dato sia almeno ripetuto una volta nella sequenza
 - un dato si considera ripetuto se, nella sequenza, se ne trova almeno un altro tale che la loro differenza, in valore assoluto, sia inferiore a 1%

■ Soluzione:

- analizzare i dati, mediante una doppia iterazione, verificando, per ognuno, che ne esista almeno uno uguale (differenza $\leq 1\%$ rispetto al massimo tra i due in valore assoluto)

■ Struttura dati:

- vettore per contenere i dati letti da file
- variabili scalari: indici, contatore e flag

■ Algoritmo:

- acquisizione dati su vettore statico sovradimensionato
- verifica mediante doppia iterazione
- quantificazione con uso di flag.

Codice

```
int datiRipetuti (FILE *fp) {
    float dati[MAXDATI];
    int ndati, i, j, ripetuto;
    fscanf(fp, "%d", &ndati);
    for (i=0; i<ndati; i++)
        fscanf(fp, "%f", &dati[i]);
    for (i=0; i<ndati; i++) {
        ripetuto = 0;
        for (j=0; j<ndati; j++)
            if (i!=j && simili(dati[i], dati[j]))
                ripetuto=1;
        if (!ripetuto) return 0;
    }
    return 1;
}
```

```
int simili (float a, float b) {
    if (fabs(a)>fabs(b))
        return (fabs(a-b)/fabs(a) < 0.01);
    else
        return (fabs(a-b)/fabs(b) < 0.01);
}
```

Selezione di dati

- Contestualmente alla verifica di più dati (o sequenze/insiemi) di dati, è possibile discriminare i dati (o il dato) che corrispondono al criterio di verifica rispetto agli altri
- La selezione può essere vista come una variante della verifica:
 - i dati vengono dapprima verificati
 - quelli (o quello) che corrispondono al criterio di accettazione vengono scelti
- La ricerca è un caso particolare di selezione:
 - si seleziona il dato (se esiste) che corrisponde al criterio di ricerca.

Conversione matricola→nome

- Formulazione:

- si realizzi una funzione C in grado di determinare il nome di uno studente, a partire dal numero di matricola (primo parametro alla funzione)
 - Siccome i numeri di matricola sono grandi (6 cifre decimali, MMAX) non si vuole sfruttare la corrispondenza indice-dato in un vettore. Le matricole sono rappresentate mediante stringhe. Si suppone che il nome abbia lunghezza massima NMAX.
- La tabella di conversione, secondo parametro alla funzione, è un vettore di `struct`, aventi come campi (stringhe) numero di matricola e nome. Il terzo parametro (intero) è la dimensione della tabella.
- Il terzo parametro è la stringa (vettore di caratteri) in cui porre il risultato

■ Soluzione:

- analizzare iterativamente i dati nel vettore, confrontando di volta in volta la matricola corrente con quella richiesta

■ Struttura dati:

- la tabella è un vettore (fornito come parametro)
- il numero di matricola e il risultato (parametri) sono stringhe

■ Algoritmo:

- la ricerca consiste in una verifica dei dati
- la funzione ricerca la matricola e ritorna il nome corrispondente.
- Il valore intero ritornato è 1 o 0 (vero o falso) indica successo o no

Codice

```
#include <string.h>
typedef struct {
    char matricola[MMAX+1], nome[NMAX+1];
} t_stud;
...
int matrNome(char m[], t_stud tabella[], int n, char n[]){
    int I;
    for (i=0; i<n; i++) {
        if (strcmp(m,tabella[i].matricola)==0) {
            strcpy (n, tabella[i].nome);
            return 1;
        }
    }
    return 0;
}
```

Codice

```
#include <string.h>
typedef struct studente {
    char matricola[MMAX+1], nome[NMAX+1];
} t_stud;
...
int matrNome(char m[], t_stud tabella[], int n, char n[]){
    int I;
    for (i=0; i<n; i++) {
        if (strcmp(m, tabella[i].matricola)==0) {
            strcpy (n, tabella[i].nome);
            return 1;
        }
    }
    return 0;
}
```

Si noti typedef per definire un sinonimo (**t_stud**) di **struct studente**.
Typedef è l'equivalente (applicato ai tipi) di #define per le costanti (es. numeri)

Problemi di ordinamento

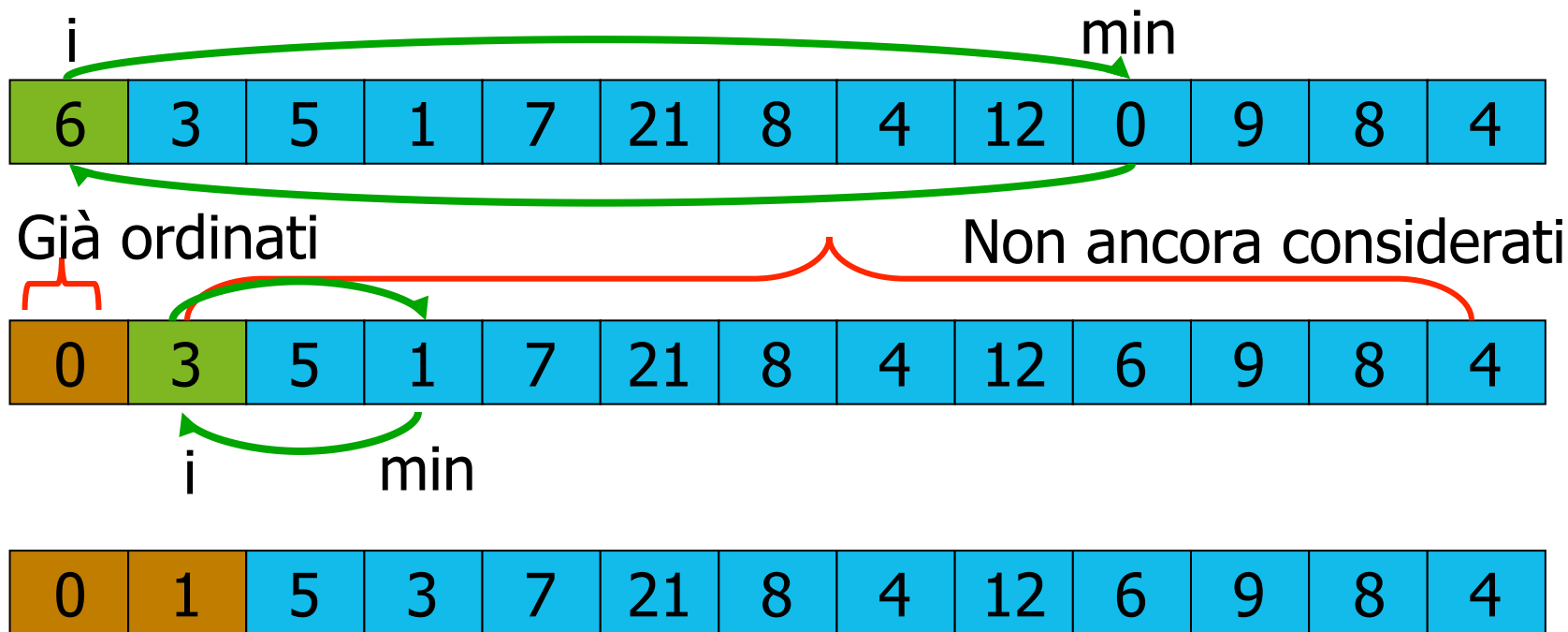
- Un problema di ordinamento consiste nella richiesta di permutare una sequenza di dati, in modo tale che (dopo la permutazione) sia verificato un criterio di ordinamento
- Per ordinare dei dati in modo totale, si opera molto spesso su un vettore, adatto a fornire una successione lineare di dati, con valori crescenti (o decrescenti) secondo la progressione crescente degli indici.

Selection sort

- Formulazione: si scriva una funzione C che:
 - ricevuti come parametri un vettore di numeri interi e la sua dimensione
 - ordini i dati in modo crescente con l'algoritmo di selection sort
- Soluzione: selection sort
 - algoritmo di ordinamento basato su ripetute ricerche/selezioni del minimo

- Struttura dati e algoritmo:
 - dati: vettore A di N interi ($A[0] \dots A[N-1]$), diviso in 2 sotto-vettori:
 - di sinistra: ordinato
 - di destra: disordinato
 - un vettore di un solo elemento è ordinato
 - approccio incrementale:
 - passo i: il minimo del sotto-vettore ($A[i] \dots A[N-1]$) è assegnato a $A[i]$; incremento di i
 - Terminazione: tutti gli elementi inseriti ordinatamente.

Esempio



Codice

```
void selectionSort (int A[], int N) {  
    int i, j, imin, temp;  
  
    for (i=0; i<N-1; i++) {  
        /*cerca indice del minimo in A[i]..A[N-1]*/  
        imin = i;  
        for (j = i+1; j < N; j++)  
            if (A[j] < A[imin]) imin = j;  
        /*scambia minimo con A[i]*/  
        temp = A[i];  
        A[i] = A[imin];  
        A[imin] = temp;  
    }  
}
```



Codice

```
void selectionSort (int A[], int N) {  
    int i, j, imin, temp;  
  
    for (i=0; i<N-1; i++) {  
        /*cerca indice del minimo in A[i]..A[N-1]*/  
        imin = i;  
        for (j = i+1; j < N; j++)  
            if (A[j] < A[imin]) imin = j;  
        /*scambia minimo con A[i]*/  
        temp = A[i];  
        A[i] = A[imin];  
        A[imin] = temp;  
    }  
}
```

Iterazione esterna, eseguita N-1 volte

Codice

```
void selectionSort (int A[], int N) {  
    int i, j, imin, temp;  
  
    for (i=0; i<N-1; i++) {  
        /*cerca indice del minimo in A[i]..A[N-1]*/  
        imin = i;  
        for (j = i+1; j < N; j++)  
            if (A[j] < A[imin]) imin = j;  
        /*scambia minimo con A[i]*/  
        temp = A[i];  
        A[i] = A[imin];  
        A[imin] = temp;  
    }  
}
```



Iterazione interna, eseguita n-i-1 volte

Codice

```
void selectionSort (int A[], int N) {  
    int i, j, imin, temp;  
  
    for (i=0; i<N-1; i++) {  
        /*cerca indice del minimo in A[i]..A[N-1]*/  
        imin = i;  
        for (j = i+1; j < N; j++)  
            if (A[j] < A[imin]) imin = j;  
        /*scambia minimo con A[i]*/  
        temp = A[i];  
        A[i] = A[imin];  
        A[imin] = temp;  
    }  
}
```

Algoritmo **"in loco"**, perché scambia i dati sul vettore (non serve un secondo vettore "di appoggio")

Sorting applicato a vettore di struct

- Esempio: funzione `ordinaStudenti` indicata, ma NON ancora realizzata
- Uno dei campi usato come chiave di ordinamento (confronto)
 - `struct studente`, campo `media`
- Conveniente realizzare funzione di confronto
 - `STUDlt` (less than), oppure `STUDge` (greater or equal), ...

```

void ordinaStudenti(struct studente el[], int n) {
    int i, j, imin;
    struct studente temp;

    for (i=0; i<n-1; i++) {
        /*cerca indice del minimo in el[i]..el[n-1]*/
        imin = i;
        for (j = i+1; j < N; j++)
            if (STUDlt(el[j],el[imin])) imin = j;
        /*scambia minimo con el[i]*/
        temp = el[i];
        el[i] = el[imin];
        el[imin] = temp;
    }
}

```

```

/* confronto: ritorna vero (non 0) se la media di s1
   e' inferiore a quella di s2, falso (0) se non lo e'
   */
int STUDlt (struct studente s1, struct studente s2) {
    return (s1.media < s2.media);
}

```

Sorting applicato a vettore di stringhe

- Un vettore di stringhe è una matrice di char
- Per gestire la matrice di char come vettore occorre
 - Usare il primo indice per identificare le righe
 - Usare la funzione strcmp per confrontare righe/stringhe
 - Usare la funzione strcpy per assegnare/copiare stringhe

```
void ordinaNomi(char nomi[][MAXL], int n) {  
    int i, j, imin;  
    char temp[MAXL];  
  
    for (i=0; i<n-1; i++) {  
        /*cerca indice del minimo in nomi[i]..nomi[n-1]*/  
        imin = i;  
        for (j = i+1; j < N; j++)  
            if (strcmp(nomi[j],nomi[imin])<0) imin = j;  
        /*scambia minimo con el[i]*/  
        strcpy(temp,nomi[i]);  
        strcpy(nomi[i],nomi[imin]);  
        strcpy(nomi[imin],temp);  
    }  
}
```

```

void ordinaNomi(char nomi[][MAXL], int n) {
    int i, j, imin;
    char temp[MAXL];

    for (i=0; i<n-1; i++) {
        /*cerca indice del minimo [i+1]*
        imin = i;
        for (j = i+1; j<n; j++)
            if (strcmp(nomi[i], nomi[j]) > 0)
                imin = j;
        /*scambia minimo con primo
        strcpy(temp, nomi[i]);
        strcpy(nomi[i], nomi[imin]);
        strcpy(nomi[imin], temp);
    }
}

```

La funzione non ha bisogno di sapere quante sono le righe:
Il programma chiamante DEVE conoscere la dimensione (il
numero di righe).

Vantaggio: la funzione può essere chiamata su matrici di
dimensioni diverse (VALE SOLO PER IL NUMERO DI RIGHE)


```

void ordinaNomi(char nomi[][MAXL], int n) {
    int i, j, imin;
    char temp[MAXL];

    for (i=0; i<n-1; i++) {
        /*cerca indice del minimo in riga i*/
        imin = i;
        for (j = i+1; j < n; j++)
            if (strcmp(nomi[j], nomi[imin]) < 0)
                imin = j;
        /*scambia minimo con elemento in posizione i*/
        strcpy(temp, nomi[i]);
        strcpy(nomi[i], nomi[imin]);
        strcpy(nomi[imin], temp);
    }
}

```

La funzione ha bisogno di sapere quante sono le righe effettivamente usate (possono essere meno del totale) per fare l'ordinamento.

Tocca al programma chiamante passare questa informazione aggiuntiva come argomento

Sul libro

- Problemi numerici:

- crivello di Eratostene (numeri primi)

- Problemi di codifica:

- cifrario di Vigenère

- Problemi di verifica:

- verifica di primalità

- Esercizi risolti:

- prodotto matrici, somma in base B, cruciverba, eliminazione di spazi, eliminazione di valori nulli, bubble sort

- Esercizi proposti