Tipi di dato strutturati: Array

- I tipi di dato visti finora sono tutti semplici: `int`, `char`, `float`, ...
- ma i dati manipolati nelle applicazioni reali sono spesso complessi (o strutturati)
- Gli array sono uno dei tipi di dato strutturati
  - sono composti da elementi omogenei (tutti dello stesso tipo)
  - ogni elemento è identificato all'interno dell'array da un numero d'ordine detto indice dell'elemento
  - il numero di elementi dell'array è detto lunghezza (o dimensione) dell'array
- Consentono di rappresentare tabelle, matrici, matrici n-dimensionali, ...

Array monodimensionali (o vettori)

- Supponiamo di dover rappresentare e manipolare la classifica di un campionato cui partecipano 16 squadre.
- È del tutto naturale pensare ad una tabella

Classifica

Squadra A	Squadra B	...	Squadra C
1^o posto	2^o posto		16^o posto

che evolve con il procedere del campionato

Classifica

Squadra B	Squadra A	...	Squadra C
1^o posto	2^o posto		16^o posto

Sintassi: dichiarazione di variabile di tipo vettore

```
tipo-elementi nome-array [lunghezza];
```

Esempio: `int vet[6];`

dichiara un vettore di 6 elementi, ciascuno di tipo intero.

- All'atto di questa dichiarazione vengono riservate (allocate) 6 locazioni di memoria consecutive, ciascuna contenente un intero. 6 è la lunghezza del vettore.
- La lunghezza di un vettore deve essere costante (nota a tempo di compilazione nel C standard, piu' avanti vedremo i VLA del C99).
- Ogni elemento del vettore è una variabile identificata dal nome del vettore e da un indice

  Sintassi: elemento di array `nome-array[espressione];`

  Attenzione: `espressione` deve essere di tipo intero ed il suo valore deve essere compreso tra 0 a lunghezza-1.

- **Esempio:**

indice	elemento	variabile
0	?	`vet[0]`
1	?	`vet[1]`
2	?	`vet[2]`
3	?	`vet[3]`
4	?	`vet[4]`
5	?	`vet[5]`

- `vet[i]` è l'elemento del vettore `vet` di indice `i`.
  Ogni elemento del vettore è una variabile.

```
int vet[6], a;
vet[0] = 15;
a = vet[0];
vet[1] = vet[0] + a;
printf("%d", vet[0] + vet[1]);
```

- `vet[0]`, `vet[1]`, ecc. sono variabili intere come tutte le altre e dunque possono stare a sinistra dell'assegnamento (es. `vet[0] = 15`), cosi' come all'interno di espressioni (es. `vet[0] + a`).
- Come detto, l'indice del vettore è un'espressione.

```
index = 2;
vet[index+1] = 23;
```

Manipolazione di vettori

- avviene solitamente attraverso cicli **for**
- l'indice del ciclo varia in genere da 0 a lunghezza-1
- spesso conviene definire la lunghezza come una costante attraverso la direttiva `#define`

Esempio: Lettura e stampa di un vettore.

```
#include <stdio.h>
#define LUNG 5

int main ()
{
int v[LUNG]; /* vettore di LUNG elementi, indicizzati da 0 a LUNG-1 */
int i;

for (i = 0; i < LUNG; i++) {
  printf("Inserisci l'elemento di indice %d:  ", i);
  scanf("%d", &v[i]);
}
printf("Indice Elemento\n");
for (i = 0; i < LUNG; i++) {
  printf("%6d %8d\n", i, v[i]); }
return 0;

}
```

Inizializzazione di vettori

- Gli elementi del vettore possono essere inizializzati con valori costanti (valutabili a tempo di compilazione) contestualmente alla dichiarazione del vettore .

  Esempio: `int n[4] = {11, 22, 33, 44};`

- l'inizializzazione deve essere contestuale alla dichiarazione

  Esempio:
  ```
  int n[4];
  n = {11, 22, 33, 44};
  ```
  $\Longrightarrow$ errore!

- se i valori iniziali sono meno degli elementi, i rimanenti vengono posti a 0

`int n[10] = {3};`	azzera i rimanenti 9 elementi del vettore
`float af[5] = {0.0};`	pone a 0.0 i 5 elementi
`int x[5] = {};`	errore!

- se ci sono più inizializzatori di elementi, si ha un errore a tempo di compilazione

  Esempio: `int v[2] = {1, 2, 3};` errore!

- se si mette una sequenza di valori iniziali, si può omettere la lunghezza (viene presa la lunghezza della sequenza)

  Esempio: `int n[] = {1, 2, 3};` equivale a

  `int n[3] = {1, 2, 3};`

- In C l'unica operazione possibile sugli array è l'accesso ai singoli elementi.
- Ad esempio, non si possono effettuare direttamente delle assegnazioni tra vettori.

Esempio:

```
int a[3] = {11, 22, 33};
int b[3];
b = a;                          errore!
```

Esempi

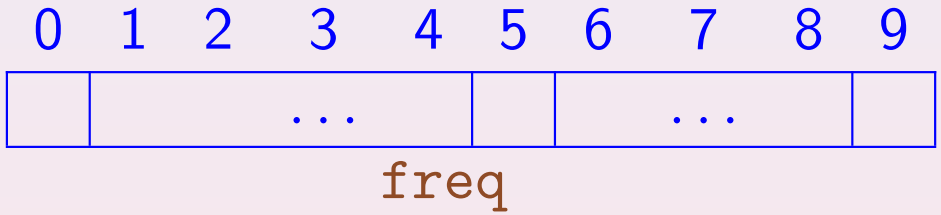
- Calcolo della somma degli elementi di un vettore.

```
int a[10], i, somma = 0;
...
for (i = 0; i < 10; i++)
  somma += a[i];
printf("%d", somma);
```

- Leggere N interi e stampare i valori maggiori di un valore intero y letto in input.

```
#include <stdio.h>
#define N 4
int main()   {
int ris[N];
int y, i;
printf("Inserire i %d valori:\n", N);
for (i = 0; i < N; i++) {
  printf("Inserire valore n.  %d:  ", i+1);
  scanf("%d", &ris[i]);     }
printf("Inserire il valore y:\n");
scanf("%d", &y);
printf("Stampa i valori maggiori di %d:\n", y);
for (i = 0; i < N; i++)
  if (ris[i] > y)
    printf("L'elemento %d:  %d e' maggiore di %d\n",
            i+1, ris[i],y);
return 0;
}
```

- Leggere una sequenza di caratteri terminata dal carattere `\n` di fine linea e stampare le frequenze delle cifre da `'0'` a `'9'`.
- utilizziamo un vettore `freq` di 10 elementi nel quale memorizziamo le frequenze dei caratteri da `'0'` a `'9'`



`freq[0]` conta il numero di occorrenze di `'0'`

...

`freq[9]` conta il numero di occorrenze di `'9'`

- utilizziamo un ciclo per l'acquisizione dei caratteri in cui aggiorniamo una delle posizioni dell'array tutte le volte che il carattere letto è una cifra

```
int i; char ch;
int freq[10] = {0};
do {
  ch = getchar();
  switch (ch) {
    case '0':  freq[0]++; break;
    case '1':  freq[1]++; break;
    case '2':  freq[2]++; break;
    case '3':  freq[3]++; break;
    case '4':  freq[4]++; break;
    case '5':  freq[5]++; break;
    case '6':  freq[6]++; break;
    case '7':  freq[7]++; break;
    case '8':  freq[8]++; break;
    case '9':  freq[9]++; break;
    }
} while (ch != '\n');
printf("Le frequenze sono:\n");
for (i = 0; i < 10; i++)

  printf("Freq.  di %d:  %d\n", i, freq[i]);
```

- Nel ciclo **do-while**, il comando `switch` può essere rimpiazzato da un **if** come segue

```
if (ch >= '0' && ch <= '9')
  freq[ch - '0']++;
```

  Infatti:

- i codici dei caratteri da `'0'` a `'9'` sono consecutivi
- dato un carattere `ch`, l'espressione intera `ch - '0'` è la distanza del codice di `ch` dal codice del carattere `'0'`. In particolare:
  - `'0' - '0' = 0`
  - `'1' - '0' = 1`
  - ...
  - `'9' - '0' = 9`

- Leggere da tastiera i risultati di 20 esperimenti. Stampare il numero d'ordine ed il valore di quelli minori del 50% della media.

```
#include <stdio.h>
#define DIM 20
int main() {
double ris[DIM], media;
int i;
/* inserimento dei valori */
printf("Inserire i %d risultati dell'esperimento:\n", DIM);
for (i = 0; i < DIM; i++) {
  printf("Inserire risultato n.  %d:  ", i);
  scanf("%g", &ris[i]); }
/* calcolo della media */
media = 0.0;
for (i = 0; i < DIM; i++)
    media = media + ris[i];
media = media/DIM;
printf("Valore medio:  %g\n", media);
/* stampa dei valori minori di media*0.5 */
printf("Stampa dei valori minori di media*0.5:\n");
for (i = 0; i < DIM; i++)
    if (ris[i] < media * 0.5)
      printf("Risultato n.  %d:  %g\n", i, ris[i]);
return 0; }
```

Array multidimensionali

Sintassi: dichiarazione

`tipo-elementi nome-array` $[\text{lung}_1][\text{lung}_2]\cdots[\text{lung}_n]$;

Esempio: **int** `mat[3][4];` $\Longrightarrow$ matrice 3×4

- Per ogni dimensione i l'indice va da 0 a lung_i-1.

		colonne			
		0	1	2	3
righe	0	?	?	?	?
	1	?	?	?	?
	2	?	?	?	?

Esempio: **int** `marketing[10][5][12]`

(indici potrebbero rappresentare: prodotti, venditori, mesi dell'anno)

Accesso agli elementi di una matrice

```
int i, mat[3][4];
...
i = mat[0][0];          elemento di riga 0 e colonna 0 (primo elemento)
mat[2][3] = 28;         elemento di riga 2 e colonna 3 (ultimo elemento)
mat[2][1] = mat[0][0] * mat[1][3];
```

- Come per i vettori, l'unica operazione possibile sulle matrici è l'accesso agli elementi tramite l'operatore `[]`.

Esempio: Lettura e stampa di una matrice.

```
#include <stdio.h>
#define RIG 2
#define COL 3
main()
{
int mat[RIG][COL];
int i, j;
/* lettura matrice */
printf("Lettura matrice %d x %d;\n", RIG, COL);
for (i = 0; i < RIG; i++)
  for (j = 0; j < COL; j++)
    scanf("%d", &mat[i][j]);
/* stampa matrice */
printf("La matrice e':\n");
for (i = 0; i < RIG; i++) {
   for (j = 0; j < COL; j++)
     printf("%6d ", mat[i][j]);
   printf("\n");      }   /* a capo dopo ogni riga */
}
```

Esempio: Programma che legge due matrici M×N (ad esempio 4×3) e calcola la matrice somma.

```
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++)
    c[i][j] = a[i][j] + b[i][j];
```

Inizializzazione di matrici

```
int mat[2][3] = {{1,2,3}, {4,5,6}};
int mat[2][3] = {1,2,3,4,5,6};
```

1	2	3
4	5	6

```
int mat[2][3] = {{1,2,3}};
int mat[2][3] = {1,2,3};
```

1	2	3
0	0	0

```
int mat[2][3] = {{1}, {2,3}};
```

1	0	0
2	3	0

Esercizio

Programma che legge una matrice A (M×P) ed una matrice B (P×N) e calcola la matrice C prodotto di A e B

- La matrice C è di dimensione M×N.
- Il generico elemento C_{ij} di C è dato da:

$$C_{ij} = \sum_{k=0}^{P-1} A_{ik} \cdot B_{kj}$$

Soluzione

```
#define M 3
#define P 4
#define N 2
int a[M][P], b[P][N], c[M][N];
...
/* calcolo prodotto */
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++) {
    c[i][j] = 0;
    for (k = 0; k < P; k++)
      c[i][j] = c[i][j] + a[i][k] * b[k][j];
  }
```

- Tutti gli elementi di `c` possono essere inizializzati a 0 al momento della dichiarazione:

```
int a[M][P], b[P][N], c[M][N] = {0};
...
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++)
    for (k = 0; k < P; k++)
      c[i][j] += a[i][k] * b[k][j];
```

Schemi di programma: ricerca e verifica

- Molti problemi riguardano la ricerca di elementi in intervalli o la verifica di proprietà.
- Sviluppiamo schemi di programma dimostrabilmente corretti che realizzano la ricerca e la verifica.
- La soluzione di problemi concreti consiste poi nella sostituzione di alcuni parametri degli schemi con valori specifici dei problemi in esame.
- Distinguiamo due tipi di ricerca: ricerca certa e ricerca incerta.
  - **ricerca certa**: si vuole determinare il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$, sapendo che almeno un elemento dell'intervallo soddisfa $\mathcal{P}$.
  - **ricerca incerta**: si vuole determinare, se esiste, il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$.

Ricerca certa

- Intervallo di ricerca: [a,b)
- Proprietà: $\mathcal{P}(\cdot)$
- Ipotesi di certezza: $\exists$ i $\in$ [a,b) . $\mathcal{P}$(i)
- Stato finale:
  x = min { i$\in$ [a,b) | $\mathcal{P}$(i)}.
- Lo schema generale per risolvere il problema è il seguente:

```
int x;
x=a;
while (!P(x))
  x=x+1;
```

- Nota: l'estremo destro dell'intervallo non serve.
- Si assume che la proprietà $\mathcal{P}$ sia esprimibile nel linguaggio.

- Proprietà invariante del ciclo:
  $x \in [a, b) \wedge (\forall\, j \in [a,x).\ \neg\, \mathcal{P}(j))$
- In altre parole, tutti gli elementi che precedono il valore corrente di x non soddisfano la proprietà $\mathcal{P}$.
- Se il ciclo termina, all'uscita dal ciclo vale la congiunzione
  $x \in [a, b) \wedge (\forall\, j \in [a,x).\ \neg\, \mathcal{P}(j)) \ \wedge\ \mathcal{P}(x)$
  che implica esattamente quanto espresso dallo stato finale:
  $x = \min\ \{\, i \in [a,b) \mid \mathcal{P}(i)\}$
- Osserviamo che l'invariante vale banalmente alla prima iterazione, con $x{=}a$.
  $a \in [a, b) \wedge (\forall\, j \in [a,a).\ \neg\, \mathcal{P}(j))$

- Verifichiamo che la proprietà
  $x \in [a, b) \wedge (\forall\, j \in [a,x).\ \neg\, \mathcal{P}(j))$
  è invariante per il ciclo:

```
x=a;
while (!P(x))
  x=x+1;
```

- Sia S uno stato in cui valgono le seguenti proprietà (x^S indica il valore di x in S)
  1. $x^S \in [a, b) \wedge (\forall\, j \in [a,x^S).\ \neg\, \mathcal{P}(j))$
  2. $\neg\, \mathcal{P}(x^S)$

  ovvero uno stato prima di una nuova iterazione del ciclo.
- 1. e 2. implicano ovviamente
  $(\forall\, j \in [a,x^S+1).\ \neg\, \mathcal{P}(j))$
- Se riusciamo anche a dimostrare che
  $x^S + 1 \in [a, b)$
  abbiamo dimostrato che la proprietà è invariante, dal momento che $x^S + 1$ è proprio il valore di x dopo la nuova iterazione.

- Sappiamo:
  $x^S \in [a, b)$
- Supponiamo per assurdo
  $x^S + 1 \notin [a, b)$ ovvero $x^S + 1 = b$ (*)
- Abbiamo appena dimostrato
  $(\forall\, j \in [a, x^S+1).\ \neg\, \mathcal{P}(j))$
  che insieme con (*) implica
  $(\forall\, j \in [a, b).\ \neg\, \mathcal{P}(j))$
- Ciò contraddice l'ipotesi di certezza
  $\exists\, i \in [a, b)\ .\ \mathcal{P}(i)$
- Dunque, dopo la nuova iterazione vale ancora la proprietà invariante.

Funzione di terminazione: Tra le tante . . . $b-x$

Ricerca certa: esempio 1

- Calcolare la radice intera di un numero naturale.
- Si può esprimere come problema di ricerca certa:
  $\lfloor\sqrt{N}\rfloor = \min\{x \in [0,N+1) \mid x^2 \leq N < (x+1)^2\}$
- Dunque l'estremo sinistro dell'intervallo di ricerca, a nello schema, in questo caso è 0, mentre l'estremo destro, b nello schema, è N.
- Infine la proprietà $\mathcal{P}(x)$ dello schema è $N < (x+1)^2$

```
int x;
x=0;
while ((x+1)*(x+1) <= N)
  x=x+1;
```

Ricerca certa: esempio 2

- Determinare la posizione della prima occorrenza di un dato elemento in un array, sapendo che tale elemento vi occorrre almeno una volta.
- Indichiamo con vet l'array e con DIM la sua dimensione
- Vogliamo determinare:
  $x = \min\{i \in [0,DIM) \mid vet[i] = el\}$
- Possiamo istanziare lo schema come segue:

```
int x;
x=0;
while (vet[x]!=el)
  x=x+1;
```

Ricerca Incerta

- Si vuole determinare, se esiste, il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$.
- Perché lo schema di ricerca certa non va bene?

```
x=a;
while (!P(x))
  x=x+1;
```

- Se l'elemento non c'è si vanno ad esaminare valori di x che sono al di fuori dell'intervallo di ricerca e per i quali la la proprietà $\mathcal{P}$ potrebbe addirittura non essere definita (errore a tempo di esecuzione).

  Esempio: Nel caso della ricerca incerta di un elemento in un array di dimensione `DIM` si andrebbero ad esaminare elementi del tipo `vet[x]` con `x > DIM`.

- Abbiamo bisogno di modificare lo schema in modo che l'analisi degli elementi avvenga solo all'interno dell'intervallo di ricerca e che la ricerca venga interrotta una volta esaurito l'intervallo (e non individuato alcun elemento).

Ricerca incerta

- Intervallo di ricerca: [a,b)
- Proprietà: $\mathcal{P}(\cdot)$
- Stato finale: $x = \min\{i \in [a,b) \mid \mathcal{P}(i)\}$ *min* b

  $\Longrightarrow$ dobbiamo stabilire quale valore calcolare se nessun elemento dell'intervallo soddisfa $\mathcal{P}$: una buona scelta è il valore b, che sicuramente non fa parte dell'intervallo.

Ricerca incerta

- Utilizziamo una variabile booleana `trovato` che fa da sentinella

  $\Longrightarrow$ impone l'uscita dal ciclo non appena si individua un elemento che soddisfa la proprietà
- in congiunzione con la sentinella, la guardia del ciclo assicura che l'intervallo di ricerca non sia esaurito

```
int trovato = FALSE; /* inizialmente false */
int x=a;
while (!trovato && x<b)
  if (P(x))
    trovato = TRUE; /*x soddisfa P */
  else
    x=x+1;
```

- Si suppone che le costanti `TRUE` e `FALSE` siano state definite opportunamente, ad esempio mediante le direttive

```
#define FALSE 0
#define TRUE 1
```

- Anche in questo caso possiamo stabilire una proprietà invariante del ciclo, questa volta un po' più complicata:

$$x \in [a,b] \ \wedge \ (\forall\, j \in [a,x).\ \neg \mathcal{P}(j)) \ \wedge \ trovato \Rightarrow \mathcal{P}(x)$$

- È facile vedere che i valori iniziali di x e trovato soddisfano banalmente l'invariante
- Inoltre, al termine del ciclo abbiamo due casi:
  1. trovato $=$ TRUE $\wedge$ x$<$b: l'invariante e questa condizione implicano $x \in [a,b) \ \wedge \ (\forall\, j \in [a,x).\ \neg \mathcal{P}(j)) \ \wedge \ \mathcal{P}(x)$
  2. trovato$=$FALSE $\wedge$ x $\geq$ b: l'invariante e questa condizione implicano $x = b \ \wedge \ (\forall\, j \in [a,b).\ \neg \mathcal{P}(j))$
- Dunque possiamo controllare l'esito della ricerca analizzando il valore di trovato

- La dimostrazione formale di invarianza della proprietà vista è lasciata per esercizio
- **Funzione di terminazione**: anche in questo caso qualcosa del tipo
  $$b - x$$
  sembra ragionevole.
- Il problema (formale) è che in un solo caso il valore di x non cresce (e dunque $b-x$ non decresce) strettamente.
- L'individuazione di una corretta funzione di terminazione è lasciata per esercizio.

Ricerca incerta: esempio

- Determinare la prima occorrenza di un elemento in un array.
- È un problema di ricerca incerta:
  $\min \{x \in [0,\text{DIM}) \mid \text{vet}[x] = \text{el}\}$ *min DIM*

```
int trovato = FALSE;
int x=0;
while (!trovato && x<DIM)
  if (vet[x]==el)
    trovato = TRUE;
  else
    x=x+1;
```

- Vi sono situazioni in cui la proprietà $\mathcal{P}$ della ricerca (certa o incerta) non è direttamente esprimibile nel linguaggio.

  Esempio: Determinare (se c'è) la posizione del primo elemento di un array di interi che è uguale alla somma degli elementi che lo precedono.
- Si tratta di un problema di ricerca incerta in cui
  1. l'intervallo [a,b] è [0,DIM]
  2. la proprietà $\mathcal{P}(x)$ è

$$vet[x] = \sum_{j=0}^{x-1} vet[j]$$

```
int trovato = FALSE;
int x=0;
while (!trovato && x<DIM)
  if (vet[x]==vet[0]+...+vet[x-1])
    trovato = TRUE;
  else
    x=x+1;
```

- In questi casi si utilizza la seguente tecnica:
  1. si rimpiazzano le espressioni "critiche" con variabili
  2. si impone l'uguaglianza tra le variabili cosi' introdotte e le corrispondenti espressioni "critiche", aggiungendo quanto necessario al corpo del ciclo per mantenere vere tali uguaglianze
- si noti che formalmente 2. corrisponde a rafforzare opportunamente l'invariante.
- Nell'esempio:

```
int trovato = FALSE;
int x=0;
int sommaPrecedenti = 0;
while (!trovato && x<DIM)
  if (vet[x]==sommaPrecedenti)
    trovato = TRUE;
  else
    { sommaPrecedenti = sommaPrecedenti + vet[x];
      x=x+1;      }
```

- Quale è l'invariante del ciclo cosi' ottenuto?

$$x \in [0,\ DIM] \wedge (\forall\ j \in [0,x).\ vet[j] \neq \sum_{k=0}^{j-1} vet[k]) \ \wedge$$

$$trovato \Rightarrow vet[x] = \sum_{k=0}^{x-1} vet[k] \wedge$$

$$sommaPrecedenti = \sum_{k=0}^{x-1} vet[k]$$

- L'ultimo congiunto rappresenta il significato della variabile introdotta per esprimere la proprietà di ricerca $\mathcal{P}$.

Verifica di una proprietà

- Vogliamo verificare che tutti gli elementi di un intervallo soddisfano una certa proprietà $\mathcal{P}$.
  1. Facciamo una ricerca incerta del minimo elemento dell'intervallo per il quale non vale la proprietà $\mathcal{P}$
  2. Se non troviamo tale minimo, la verifica ha esito positivo, altrimenti ha esito negativo.
- Lo schema generale per risolvere questo problema.

```
int trovato = FALSE;
int x=a;
while (!trovato && x<b)
  if (!P(x))
    trovato = TRUE;
  else
    x=x+1;
if (trovato)
  /* esito negativo */
else
  /* esito positivo */
```