



Capitolo 2: i tipi aggregati

DEFINIZIONE E USO DI ARRAY, MATRICI E
STRUCT IN C



Vettori e matrici

AGGREGATI DI DATI OMOGENEI ACCESSIBILI MEDIANTE
INDICI

Vettori (array)

- Aggregati di uno stesso tipo base

- Definizione (dichiarazione)

`<tipo base> <identificatore>[<dimensione>]`

- **Esempi:** `int v[10]; char s[L]; float w[N];`
- La dimensione deve essere una costante (un intero oppure una costante intera con `const int` o `#define`)
- Nella dichiarazione è possibile una inizializzazione esplicita:
 - `int v[10] = {-2,0,1,10,-7,12,34,9,-3,6};`
 - `char s1[6] = {'h','e','l','l','o','\0'};`
 - `char s2[6] = "hello"; // Equivale alla precedente`
- Ci sono regole, omesse qui, per inizializzazioni parziali e dimensioni implicite

Vettori (array)

- Le caselle del vettore sono numerate da 0 a N-1 (con N dimensione):
 - Esempi: `v[0]`, `v[1]`, ..., `v[N-1]`
 - NON ci sono operazioni atomiche su tutto un vettore:
 - Eccezioni: inizializzazione (l'assegnazione a TUTTO il vettore non può essere fatta in altre istruzioni)
 - Le stringhe (vettori di char) hanno operazioni atomiche ma sono chiamate a funzioni di libreria (`strcpy`, `strcmp`, `strcat`, `fgets`, `fputs`, ...), oppure IO formattato con `%s`
 - NON ci sono operazioni atomiche su parte (es. un intervallo di caselle) del vettore
 - Occorre accedere a una casella alla volta, come se fosse una variabile a se stante
 - **Esempi:** `v[0] = x`; `s[4] = 'a'`; `w[i] = w[j]`; `s[N-j-1] = s[0]`;

Vettori (array)

- Vettore come parametro a funzione: di fatto passato «by reference»
 - **Passo l'indirizzo del (primo elemento del) vettore**
 - Concetto: passando l'indirizzo posso modificarne il valore, il vettore viene condiviso tra funzione e programma chiamante
 - Si possono omettere le dimensioni nel parametro formale
 - Es: `int strlen(int s[])`
 - La teoria dei puntatori verrà ripresa in dettaglio più avanti

Esempio

```
int eta[20], altezza[20], i;  
float etaMedia = 0.0;  
float altezzaMedia = 0.0;  
  
for(i=0; i<20; i++) {  
    scanf("%d %d", &eta[i], &altezza[i]);  
    etaMedia += eta[i];  
    altezzaMedia += altezza[i];  
}  
etaMedia = etaMedia/20;  
altezzaMedia = altezzaMedia/20;  
  
/* altro lavoro sui vettori */
```

Matrici (vettori multidimensionali)

- Vettori a due o più dimensioni: è sufficiente usare due o più livelli di parentesi quadre
- Definizione (dichiarazione)
`<tipo base> <identificatore>[<dim1>][<dim2>]`
 - **Esempi:** `int M[10][10]; char s[R][C]; float W[N1][N2];`
 - Le dimensioni devono essere costanti (un intero oppure una costante intera)
 - Nella dichiarazione è possibile una inizializzazione esplicita:
`int v[5][2] = {{-2,0,1,10,-7},{12,34,9,-3,6}};`
`char giorni[7][9] = {"lunedì","martedì","mercoledì","giovedì",
"venerdì","sabato","domenica"};`

Matrici (vettori multidimensionali)

- Le caselle della matrice sono numerate con un indice per ogni dimensione (riga - colonna)
 - Esempi: $M[0][j]$, $s[r][c]$, ..., $w[N-1][0]$
 - NON ci sono operazioni atomiche su tutta la matrice (eccetto, come per i vettori, l'inizializzazione)
 - NON ci sono operazioni atomiche su parte (es. una sotto-matrice) della matrice
 - E' possibile identificare una sotto-matrice (es. una riga) omettendo gli ultimi indici
 - **Esempio:** (matrici bidimensionali di caratteri) : $s[r]$ (riga r , su cui è possibile fare `fgets(s[r], MAX, fin)`)
 - Occorre accedere a una casella alla volta, come se fosse una variabile a se stante
 - **Esempi:** $v[0][0] = x$; $s[4][1] = 'a'$; $w[i][j] = w[j][i]$;
- Matrice come parametro a funzione: "by reference" come i vettori
 - Si **possono** omettere le dimensioni nel parametro **formale**

Esempio

```
int matrice_diagonale[3][3] = { { 1, 0, 0 },  
                                { 0, 1, 0 },  
                                { 0, 0, 1 } } ;  
  
float M2 [N][M], V[N], Y[M];  
  
/* istruzioni che assegnano valori a M e V */  
...  
/* prodotto matrice vettore */  
for (r=0; r<N; r++) {  
    Y[r] = 0.0;  
    for (c=0; c<M; c++)  
        Y[r] = Y[r] + M2[r][c]*V[c];  
}
```

Tipi struct

AGGREGATI DI CAMPI ETEROGENEI, ACCESSIBILI
MEDIANTE NOME

I tipi `struct`

- Il dato aggregato in C è detto `struct`
 - In altri linguaggi si parla di `record`
- Una `struct` (struttura) è un dato costituito da campi:
 - I campi sono aggregati in una singola variabile
 - I campi sono di tipi (base) noti (eventualmente altre `struct` o *puntatori*)
 - Ogni campo all'interno di una `struct` è accessibile mediante un identificatore (anziché un indice, come nei vettori)
- Una definizione di `struct` equivale ad una definizione di tipo
 - Successivamente, una struttura può essere usata come un tipo per dichiarare variabili

Esempio

```
struct studente {  
    char cognome[MAX], nome[MAX];  
    int matricola;  
    float media;  
};
```

Esempio

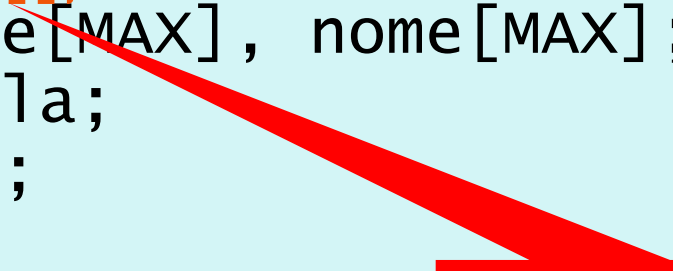
```
struct studente {  
    char cognome[MAX], nome[MAX];  
    int matricola;  
    float media;  
};
```

Nuovo tipo di
dato

- Il nuovo tipo definito è **struct studente**
- La parola chiave **struct** è obbligatoria

Esempio

```
struct studente {  
    char cognome[MAX], nome[MAX];  
    int matricola;  
    float media;  
};
```



Nome del tipo
aggregato

- Stesse regole che valgono per i nomi delle variabili
- I nomi di struct devono essere diversi da nomi di altre struct (possono essere uguali a nomi di variabili)

Esempio

```
struct studente {  
    char cognome[MAX], nome[MAX];  
    int matricola;  
    float media;  
};
```

Campi (eterogenei)

- I campi corrispondono a variabili locali di una struct
- Ogni campo è quindi caratterizzato da un tipo (base) e da un identificatore (unico per la struttura)

Schemi di dichiarazione

1. Schema base

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct studente s, t;
```


Schemi di dichiarazione

2. Dichiarazione/definizione contestuale di tipo struct e variabili

- Tipo `struct` e variabili (`s` e `t` nell'esempio) vanno definiti nello stesso contesto (globale o locale)

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} s, t;
```

Schemi di dichiarazione

3. (uso raro) Dichiarazione/definizione contestuale di tipo `struct` (senza identificatore) e variabili

- Tipo `struct` utilizzato unicamente per le variabili definite contestualmente (senza un identificativo)
- **NON si possono definire variabili dello stesso tipo in altre istruzioni dichiarative o in funzioni**

```
struct
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} s, t;
```

Typedef: la define dei tipi

- E' possibile associare un indentificatore a un tipo esistente:
 - `typedef <tipo esistente> <nuovo nome>;`
 - Esempio:
`typedef int number;`
`...
number n, m;`
- In pratica
 - Serve per associare a un tipo un nuovo nome (utile specialmente per tipi `struct`)
 - Ha una funzione simile alla `#define` per le costanti letterali
 - Typedef NON è direttiva al pre-compilatore, viene gestita dal compilatore

Schemi di dichiarazione

4. Sinonimo di `struct` `studente` introdotto mediante `typedef`

```
typedef struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} Studente;

...
Studente s, t;
```

Schemi di dichiarazione

5. Sinonimo introdotto mediante **typedef**: variante senza identificatore di **struct**

```
typedef struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} Studente;
...
Studente s, t;
```

Identificatore inutilizzato

Schemi di dichiarazione

5. Sinonimo introdotto mediante **typedef**: variante senza identificatore di **struct**

```
typedef struct
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} Studente;

...
Studente s, t;
```

Accesso ai campi di una **struct**

- Dopo aver dichiarato una variabile di tipo struct, si può accedere ai campi della variabile attraverso l'operatore '.'

<identificativo della variabile di tipo struct> . <nome del campo>

- Esempio:

```
typedef struct{  
    double re;  
    double im;  
} complex;  
  
...  
complex num1, num2;  
num1.re = 0.33; num1.im = -0.43943;  
num2.re = -0.133; num2.im = -0.49;
```

struct e vettori

- Analogia:
 - Sono entrambi tipi di dati aggregati
- Differenze:
 - Dati eterogenei (`struct`) / omogenei (vettori)
 - Accesso per nome (`struct`) / indice (vettori)
 - Parametri per valore (`struct`) / per riferimento (vettori)
 - Accesso parametrizzato non ammesso (`struct`) / ammesso (vettori)

Accesso parametrizzato a vettori

- I vettori sono frequentemente utilizzati per accesso parametrizzato a dati numerati (es. in costrutti iterativi) – con **"parametrizzato"** si intende ***"la scelta della casella dipende dalla variabile i"***

Esempio

```
for (i=0; i<N; i++) {  
    ...  
    dati[i] = ...;  
    ...  
}
```

Accesso parametrizzato a **struct**: NO

- Ai campi di una struct NON si può accedere in modo parametrizzato



```
char campo[20];  
...  
scanf("%s", campo);  
printf("%s", s.campo);
```

Accesso parametrizzato a **struct**: NO

- Ai campi di una **struct** NON si può accedere in modo parametrizzato



```
char campo[20];  
...  
scanf("%s", campo);  
printf("%s", s.campo);
```

campo è una variabile!

I campi della struct sono: cognome, nome, matricola, media
Non posso usare la variabile campo per indicare uno dei campi della struttura letto da tastiera (come invece faccio con una variabile intera, ad esempio i, per i vettori)

Accesso parametrizzato a **struct**: NO

- Soluzione: utilizzare una funzione (da realizzare) per il passaggio da identificatore (variabile) di campo a struttura



```
char campo[20];  
...  
scanf("%s", campo);  
stampaCampo(s, campo);
```

La funzione NASCONDE i dettagli

```
/* nella funzione l'accesso è esplicito, non parametrizzato */  
void stampaCampo(  
    struct studente s, char id[]) {  
    if (strcmp(id,"cognome")==0)  
        printf("%s",s.cognome);  
    else if(strcmp(id,"nome")==0)  
        printf("%s",s.nome);  
    else if(strcmp(id,"matricola")==0)  
        printf("%d",s.matricola);  
    ...  
}
```

Meglio struct o vettore?

- Per dati omogenei (es. punto come elenco delle coordinate, numero complesso, ...): meglio `p.x`, `p.y` o `p[0]`, `p[1]`?
- Meglio `struct` o vettore ?
 - Soluzione 1: `struct` consigliata se:
 - Pochi campi
 - Meglio identificarli per nome
 - Non serve accesso parametrizzato (i.e., attraverso variabile)
 - Si vuole poter trattare la `struct` come un unico dato (es. per assegnazioni a variabili, parametro unico o valore di ritorno da funzioni)
 - Allora meglio `p.x`, `p.y` !!!

Vettori come campi di `struct`

- Una `struct` può avere uno o più vettori come campi

Esempio: cognome e nome in `struct` studente

- Attenzione!

- Una `struct` viene passata per valore a una funzione, mentre un vettore sarebbe passato per riferimento

- Se una `struct` ha come campo un vettore di N elementi, passare la `struct` come parametro richiede "copiare" tutto il vettore
- *TRUCCO: avvolgere/inglobare un vettore o una matrice in una `struct` è un trucco usato se "proprio si vuole" passare il vettore o la matrice "per valore" ("by value")*

Vettore di `struct`

Spesso si usano vettori di `struct`

- Sono consigliati per collezioni (numerabili) di aggregati eterogenei
 - Esempio: gestione di un elenco di studenti
- Attenzione!
 - Un vettore (anche se di `struct`) viene passato per riferimento a una funzione
 - Una funzione può quindi modificare il contenuto di un vettore (ad esempio ordinare i dati)

Vettore di struct

```
int main(void) {  
    struct studente elenco[NMAX];  
    int i, n;  
  
    printf("quanti studenti(max %d)? ", NMAX);  
    scanf("%d", &n);  
    for (i=0; i<n; i++) {  
        elenco[i] = leggiStudente();  
    }  
    ordinaStudenti(elenco, n);  
    printf("studenti ordinati per media\n");  
    for (i=0; i<n; i++) {  
        stampaStudente(elenco[i]);  
    }  
}
```

Vettore di struct


```
int main(void) {  
    struct studente elenco[NMAX];  
    int i, n;  
  
    printf("quanti studenti(max %d)? ", NMAX);  
    scanf("%d", &n);  
    for (i=0; i<n; i++) {  
        elenco[i] = leggiStudente();  
    }  
    ordinaStudenti(elenco, n);  
    printf("studenti ordinati per media\n");  
    for (i=0; i<n; i++) {  
        stampaStudente(elenco[i]);  
    }  
}
```

vettore di struct

Vettore di struct

```
int main(void) {  
    struct studente elenco[NMAX];  
    int i, n;  
  
    printf("quanti studenti(max %d)? ", NMAX);  
    scanf("%d", &n);  
    for (i=0; i<n; i++) {  
        elenco[i] = leggiStudente();  
    }  
    ordinaStudenti(elenco, n);  
    printf("studenti ordinati per media\n");  
    for (i=0; i<n; i++) {  
        stampaStudente(elenco[i]);  
    }  
}
```

Passaggio per riferimento



Vettore di struct

```
int main(void) {
    struct studente elenco[NMAX];
    int i, n;

    printf("quanti studenti(max %d)? ",NMAX);
    scanf("%d",&n);
    for (i=0; i<n; i++) {
        elenco[i] = leggiStudente();
    }
    ordinaStudenti(elenco,n);
    printf("studenti ordinati per media\n");
    for (i=0; i<n; i++) {
        stampaStudente(elenco[i]);
    }
}
```

```
void ordinaStudenti(struct studente el[],
                    int n) {

    /*
        funzione da scrivere.
        MODIFICA il contenuto del vettore
        riordinando gli studenti per
        media crescente.

        Funzione completata alla fine del
        capitolo (col selectionSort)
    */
}
```