
Laboratorio di programmazione

Lezione VI

Tatiana Zolo

`tatiana.zolo@libero.it`

LE STRUCT

Tipo definito dall'utente i cui elementi possono essere *eterogenei* (di tipo diverso).

Introduce un **nuovo tipo** di dato.

LE STRUCT

Tipo definito dall'utente i cui elementi possono essere *eterogenei* (di tipo diverso).

Introduce un **nuovo tipo** di dato.

→ Dichiarazione:

```
struct nome_struttura {  
    tipo var1;  
    tipo var2;  
    ...  
    tipo vark;  
};
```

Ora posso creare delle variabili di tipo nome_struttura:

```
nome_struttura var_struttura;
```

LE STRUCT

→ Altra possibilità:

```
struct nome_struttura {  
    tipo var1;  
    tipo var2;  
    ...  
    tipo vark;  
} var_struttura1, var_struttura2;
```

In tal caso nome_struttura non è necessario.

Necessario se si vuole definire un'altra variabile dello stesso tipo successivamente:

```
nome_struttura var_struttura3;
```

LE STRUCT

→ Accesso ai membri: tramite l'operatore "punto".

```
nome_struttura.var1 = ...;  
tipo t = nome_struttura.var2;
```

LE STRUCT

→ **Accesso ai membri:** tramite l'operatore "punto".

```
nome_struttura.var1 = ...;  
tipo t = nome_struttura.var2;
```

→ **Assegnamento di strutture:** solo se sono dello stesso tipo.

```
struct s1 {  
    int a, b;  
}  
struct s2 {  
    int a, b;  
}  
...  
s1 x, y;  s2 z;  
...
```

`y = x;` → OK

`z = x;` → ERRORE: non corrispondenza di tipo.

LE STRUCT

- **Confronto tra strutture:** non possibile. Il seguente frammento di codice produce un errore:

```
if (x == y)
    ...
```

Occorre considerare i singoli campi delle strutture:

```
if (x.a == z.a && x.b == z.b)
    ...
```

LE STRUCT

→ **Confronto tra strutture:** non possibile. Il seguente frammento di codice produce un errore:

```
if (x == y)
    ...
```

Occorre considerare i singoli campi delle strutture:

```
if (x.a == z.a && x.b == z.b)
    ...
```

→ **n.b.**

- strutture diverse possono avere campi con lo stesso nome;
- i nomi dei campi possono essere gli stessi di variabili e funzioni già utilizzate.

ARRAY DI STRUCT

Si definisce una struttura e poi si dichiara un array di quel tipo.

Esempio:

```
struct studenti {  
    char nome[50];  
    char cognome[50];  
    int anno_immatricolazione;  
};  
studenti elenco_studenti[100];
```

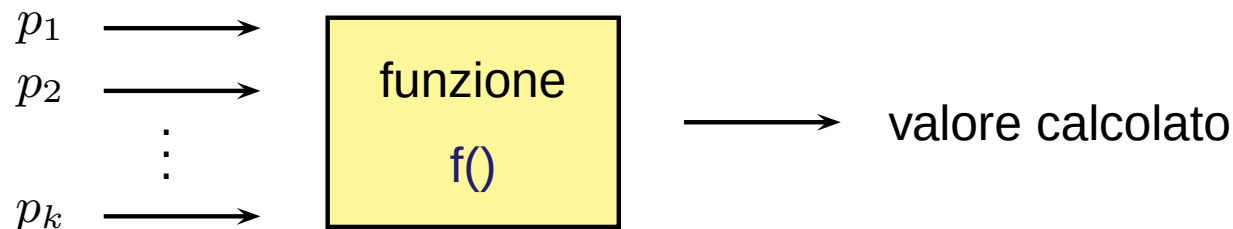
Per accedere ad una struttura specifica $\Rightarrow$ indicizzare il nome della struttura: `cout << elenco_studenti[2].cognome` stampa il cognome memorizzato nella variabile membro `cognome` dello studente i cui dati sono nella 3° struttura dell'array (l'indice dell'array parte sempre da 0!).

LE FUNZIONI

- Dobbiamo risolvere un problema complesso: come fare?
 - Suddivisione del problema in (sotto)problemi più semplici;
 - risoluzione dei problemi più semplici;
 - combinazione delle soluzioni dei problemi più semplici per ottenere la soluzione del problema di partenza.

LE FUNZIONI

- Dobbiamo risolvere un problema complesso: come fare?
 - Suddivisione del problema in (sotto)problemi più semplici;
 - risoluzione dei problemi più semplici;
 - combinazione delle soluzioni dei problemi più semplici per ottenere la soluzione del problema di partenza.
- Il programmatore può definire *sottoprogrammi* che risolvono problemi specifici: i sottoprogrammi si realizzano attraverso le **funzioni**.



dove $p_1, \dots, p_k$ sono i parametri di ingresso.

f() risolve un problema specifico e scambia informazioni con il `main()` e le altre funzioni attraverso i parametri e il valore calcolato.

LE FUNZIONI

Ogni funzione è composta da 4 parti (che insieme costituiscono la **definizione della funzione**):

1. tipo di ritorno;
2. nome della funzione;
3. lista dei parametri;
4. corpo della funzione.

Le prime tre parti insieme rappresentano il **prototipo della funzione** o la **dichiarazione della funzione**.

Invocazione funzione $f()$ $\implies$ controllo del programma ad $f()$ e sospensione dell'esecuzione della funzione attiva.

Fine esecuzione di $f()$ (ultima istruzione oppure *istruzione di ritorno*) $\implies$ la funzione sospesa riprende l'esecuzione.

PROTOTIPO DI UNA FUNZIONE

1. Tipo di ritorno: può essere

- tipo predefinito (es. `int`);
- tipo composto (es. `double*`, cioè puntatori);
- tipo definito dall'utente (es. `struct`, classi);
- `void`, cioè la funzione non restituisce alcun valore.

Non si possono usare come tipi di ritorno un tipo funzione oppure un tipo array predefinito.

C++ standard: il tipo di ritorno non può essere omesso $\implies$ errore a tempo di compilazione.

PROTOTIPO DI UNA FUNZIONE

1. Tipo di ritorno: può essere

- tipo predefinito (es. `int`);
- tipo composto (es. `double*`, cioè puntatori);
- tipo definito dall'utente (es. `struct`, classi);
- `void`, cioè la funzione non restituisce alcun valore.

Non si possono usare come tipi di ritorno un tipo funzione oppure un tipo array predefinito.

C++ standard: il tipo di ritorno non può essere omesso $\implies$ errore a tempo di compilazione.

3. Lista dei parametri:

- in una definizione di funzione un nome di parametro consente di accedere al parametro nel corpo della funzione;
- in una dichiarazione di funzione il nome del parametro non è necessario (se c'è può essere diverso da quello della def. di funz.).

Controllo di tipo dei parametri al momento della compilazione.

PASSAGGIO DEGLI ARGOMENTI

→ Passaggio per valore:

```
int nome_funz(int n) {  
    n = n + 1;  
    return n;  
}
```

Chiamata: $n = 0$; $m = \text{nome_funz}(n)$;

Risultato: $n = 0$, $m = 1$

→ Passaggio per riferimento:

```
int nome_funz(int& n) {  
    n = n + 1;  
    return n;  
}
```

Chiamata: $n = 0$; $m = \text{nome_funz}(n)$;

Risultato: $n = 1$, $m = 1$

PARAMETRI ARRAY E MATRICI

- **Array monodimensionali.** In C++ un array non è mai passato per valore, ma come un puntatore al suo primo elemento. Conseguenze:
 - le modifiche sono effettuate sull'array stesso e non su una copia locale;
 - la dimensione di un array non fa parte del suo tipo parametro.

Esempio di definizione di una funzione avente come parametro un array:

```
int max(int v[], int dim) { ... }
```

Esempio di chiamata della funzione max:

```
int massimo_voto = max(voti, numero_voti)
```

PARAMETRI ARRAY E MATRICI

- **Array monodimensionali.** In C++ un array non è mai passato per valore, ma come un puntatore al suo primo elemento. Conseguenze:
 - le modifiche sono effettuate sull'array stesso e non su una copia locale;
 - la dimensione di un array non fa parte del suo tipo parametro.

Esempio di definizione di una funzione avente come parametro un array:

```
int max(int v[], int dim) { ... }
```

Esempio di chiamata della funzione max:

```
int massimo_voto = max(voti, numero_voti)
```

- **Array bidimensionali (matrici).** Se utilizzato come parametro va specificato il numero di colonne. Esempio di definizione di una funzione avente come parametro una matrice:

```
void visualizza_mat(int mat[][3], int righe) { ... }
```

PARAMETRI ARRAY E MATRICI

- **Array monodimensionali.** In C++ un array non è mai passato per valore, ma come un puntatore al suo primo elemento. Conseguenze:
 - le modifiche sono effettuate sull'array stesso e non su una copia locale;
 - la dimensione di un array non fa parte del suo tipo parametro.

Esempio di definizione di una funzione avente come parametro un array:

```
int max(int v[], int dim) { ... }
```

Esempio di chiamata della funzione max:

```
int massimo_voto = max(voti, numero_voti)
```

- **Array bidimensionali (matrici).** Se utilizzato come parametro va specificato il numero di colonne. Esempio di definizione di una funzione avente come parametro una matrice:

```
void visualizza_mat(int mat[][3], int righe) { ... }
```

es.array_candidati.cpp.

ESERCIZI

1. **Struct:** creare una struct "persona" (il nome sceglietelo pure voi, ma sensato) con due membri: nome e numero di telefono. Scrivere un programma che, una volta riempite con i dati necessari tre struct "persona", dia le seguenti 3 possibilit :
 1. conoscere il nome a partire da un numero di telefono;
 2. conoscere il numero di telefono a partire da un nome;
 3. uscire dal programma.(nome_telefono.cpp).
2. **Array di struct e funzioni:** leggere 'n' struct di persone ('n' inserito dall'utente) con due campi, nome ed et , e restituire il nome della piu' giovane o di una delle piu' giovani. Utilizzare una funzione per la lettura dei dati delle struct e una che restituisca l'indice della persona selezionata. (piu_giovane_array.cpp).

ESERCIZI

3. **Array di struct e funzioni:** crea un array di struct "studente" (cognome, array contenente i voti degli esami). Presenta all'utente un menu' con varie possibilita':
- media (in trentesimi);
 - media (in 110);
 - voto piu' alto;
 - voto piu' basso;
 - fine programma.

Realizza ciascuna operazione con una funzione.

Possibili estensioni dell'esercizio: altre operazioni possibili sono vedere quante volte e' stato preso un certo voto; fare un grafico dell'andamento dei voti (es. con una matrice di caratteri).

Ampliare la struct "studente" inserendo altri dati: nome materia, data esame, eventuale lode, ecc. (`voti_esami.cpp`).