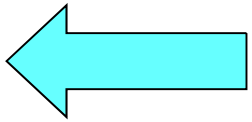
17. Stringhe e struct

Andrea Marongiu
(andrea.marongiu@unimore.it)
Paolo Valente

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Contenuto lezione

- Stringhe 
- Struct
  - Operazioni
  - Progettazione strutture dati
  - e passaggio parametri
- Matrici statiche
  - Implementazione
  - Passaggio alle funzioni

Stringa 1/2

- Una *stringa* è una sequenza di caratteri
  - Questa è la definizione di un oggetto (tipo di dato) astratto
  - Vedremo a breve come implementarlo con oggetti concreti
- Letterale stringa (costante senza nome): sequenza di caratteri che costituisce la stringa, delimitata da doppi apici
- Esempio: la costante letterale per la stringa *sono una stringa* è
- **`"sono una stringa"`**
- All'oggetto *cout* abbiamo spesso passato dei letterali di tipo stringa mediante l'operatore di uscita <<

Altri esempi costanti stringa

`"Hello\n"`

`"b"` (stringa contenente un solo carattere, per il momento la consideriamo equivalente a **`'b'`**)

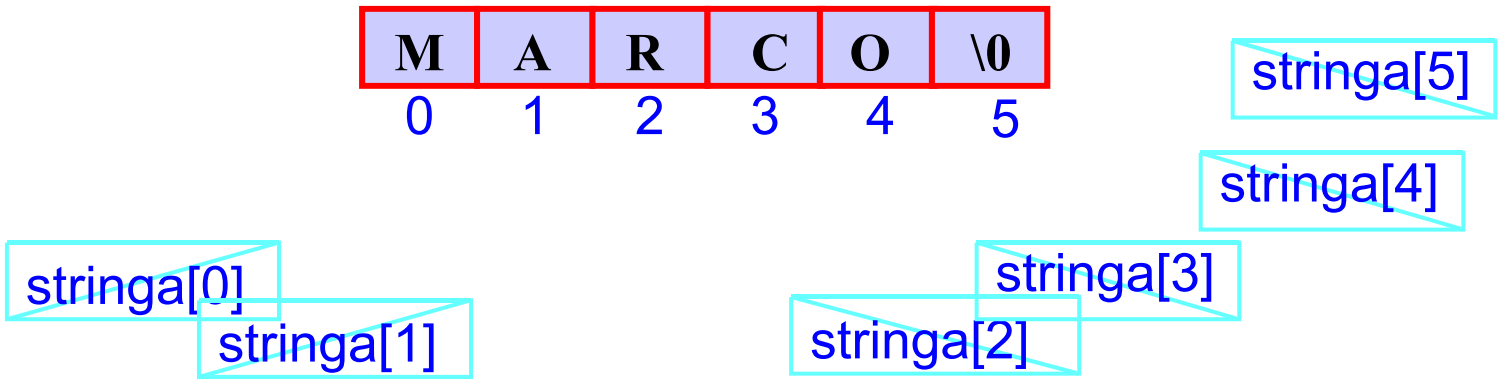
`""` **stringa nulla**

Stringhe in C/C++

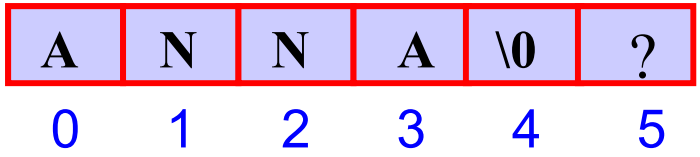
- Nel linguaggio C/C++ non esiste propriamente il tipo stringa
- E' implementato concretamente mediante
  - Un array di caratteri terminati da un carattere **terminatore**
  - Il terminatore è il carattere speciale `'\0'`
    - Numericamente, il suo valore è 0
- Come per i vettori, nella libreria standard del C++ è disponibile anche un tipo astratto stringa (*string*) con interfaccia di più alto livello di un array di caratteri
  - In questo corso non vedremo tale tipo astratto

Sintassi definizione

- SINTASSI della definizione di un oggetto di tipo stringa:
- [**const**] **char** <*identificatore*> **[** <*espr-costante*> **]** ;
- Esempio:
- **char stringa[6] ;**
  - alloca spazio per 6 oggetti di tipo **char**
  - uno va utilizzato per il fine stringa **'\0'**
  - quindi la stringa ha al più 5 caratteri



Note 1/2

- L'istruzione
- **`char stringa[N] ;`**
  - Alloca spazio per una stringa di al più *N*-1 caratteri
  - E' possibile memorizzare in un array di *N* caratteri anche una stringa di dimensione inferiore ad *N*-1
  - Esempio:
  - **`char nome[6] ;`**
- 

- in questo caso le celle oltre il carattere '\0' sono **concettualmente vuote**
  - ovviamente contengono pur sempre un valore, che però non viene preso in considerazione

Note 2/2

- Una stringa è implementata mediante un *array* di caratteri
- Ma un *array* di caratteri non è necessariamente l'implementazione di una stringa
- Affinché un *array* di caratteri implementi una stringa, è **necessario** che contenga il terminatore **`'\0'`**

Inizializzazione

- Vi sono tre modi per inizializzare una stringa in una definizione:

```
char nome[6] = { 'M', 'A', 'R', 'C', 'O', '\0' } ;
 /* come un normale array */
char nome[6] = "MARCO" ;
 /* sintassi utilizzabile solo per le stringhe; il
    carattere di fine stringa viene inserito
    automaticamente dal compilatore */

char nome[] = "MARCO" ;
 /* in questo caso la stringa viene dimensionata
    automaticamente a 6 ed il carattere di fine
    stringa viene inserito dal compilatore */
```

Assegnamento

- Se non si tratta di una inizializzazione, l'unico modo per assegnare un valore ad una stringa è carattere per carattere (come un normale array), con esplicito inserimento del carattere di fine stringa:

```
char nome[6];
nome[0]= 'M';
nome[1]= 'A';
nome[2]= 'R';
nome[3]= 'C';
nome[4]= 'O';
Nome[5]= '\0';
```

Input/output di stringhe

- Se un oggetto di tipo stringa (ossia array di caratteri)
  - viene passato al *cout*/*cerr* mediante l'operatore <<
    - vengono stampati tutti i caratteri dell'array, finché non si incontra il terminatore
  - viene utilizzato per memorizzarvi ciò che si legge da *cin* mediante l'operatore >>
    - vi finisce dentro la prossima **parola**, ossia sequenza di caratteri non separati da spazi
    - Esempio: se sullo *stdin* vi è "*ciao mondo*", nella stringa finisce solo "*ciao*" (e sullo *stdin* rimane " *mondo*")
- Esercizio: definire un oggetto di tipo stringa, inizializzarlo, stamparlo, riversarvi dentro il contenuto dello *stdin*, ristamparlo

Soluzione

```
main()
{
      const int MAX_LUN = 20 ;
      char stringa[MAX_LUN] = "prova";
      cout<<stringa<<endl ;
      cin>>stringa ;
      cout<<stringa<<endl ;
}
```

Domanda

- Che succede se l'utente immette una parola più lunga delle dimensioni massime della stringa che avete definito nel programma?

Errori

- Definire un array di caratteri ed inizializzarlo successivamente come una stringa
  - ESEMPIO DI SEQUENZA DI ISTRUZIONI ERRATA:
  - **`char nome[6];`**
  - **`nome = "MARCO" ;`** **NO**
-
- Copiare una stringa in un'altra con l'operazione di assegnamento
  - ESEMPIO DI SEQUENZA DI ISTRUZIONI ERRATA:
  - **`char nome[15], cognome[15] ;`**
  - **`nome = cognome;`** **NO**
  -
- In conclusione, come abbiamo già visto parlando degli array, gli elementi vanno copiati uno alla volta

Domanda

- C'è differenza tra `'A'` ed `"A"` ?
- Occupano lo stesso spazio in memoria?

Caratteri e stringhe

- `'A'` carattere *A*, rappresentabile in un oggetto di tipo **`char`**, ad esempio
- **`char c = ‘A’ ;`**
- **`"A"`** stringa *A*, rappresentabile in un array di due caratteri, ad esempio
- **`char s[2] = "A" ;`**
- Tale differenza ha un impatto anche sulla rappresentazione in memoria:

Stringa ed elementi

- I singoli caratteri di una stringa possono anche essere visti come oggetti indipendenti
- "MARCO" → 'M' 'A' 'R' 'C' 'O' '\0'
- Se pensati come stringa sono però parte di un tutt'uno

Stringhe statiche e dinamiche

- Nell'accezione comune, una stringa è una sequenza di caratteri la cui lunghezza può variare
  - Per supportare stringhe dinamiche di qualsiasi lunghezza bisognerebbe utilizzare l'allocazione dinamica della memoria
- Poiché tale argomento sarà trattato in seguito, per ora si prenderà in considerazione solo il caso di
  - stringhe statiche (dimensione fissa)
  - stringhe dinamiche con dimensione massima definita a tempo di scrittura del programma

Esercizio

- Scrivere un programma che legga una parola da *stdin* e ne stampi la lunghezza
  - Senza utilizzare funzioni di libreria per le stringhe
- Ripetere l'esercizio utilizzando invece una stringa contenente più parole, ed inizializzata a piacere

Idea

- Scandire tutto l'array che rappresenta la stringa fino al carattere di terminazione `'\0'`, contando i passi che si effettuano

Algoritmo e struttura dati

- Algoritmo
  - Inizializzare una variabile contatore a 0
  - Ripetere un ciclo fino al carattere `'\0'` ed incrementare la variabile contatore ad ogni passo
  - Stampare il valore finale della variabile contatore
- Struttura dati
  - Una variabile per memorizzare la stringa
  - Una variabile ausiliaria come indice del ciclo e forse un'ulteriore variabile come contatore del numero di caratteri ...

Programma

```
main()
{
  int conta=0;
  char dante[]=“Nel mezzo del cammin di nostra vita”;

  for (int i=0; dante[i]!='\0'; i++)
      conta++; // poteva bastare la sola variabile i

  cout<<“Lunghezza stringa = “<<conta<<endl ;
 }
```

Domanda

```
main()
{
  int conta=0;
  char dante[]="Ho preso 0 spaccato";

  for (int i=0; dante[i] != '\0'; i++)
      conta++;

  cout<<"Lunghezza stringa = "<<conta<<endl ;
 }
```

E' corretto?

Risposta

- Sì, perché il codice del carattere `'\0'` è diverso dal codice del carattere `'0'`

Esercizio

- Scrivere un programma che legga una parola da *stdin* e ne assegni il contenuto ad un'altra stringa
  - la stringa di destinazione deve essere memorizzata in un *array* di dimensioni sufficienti a contenere la stringa sorgente
  - il precedente contenuto della stringa di destinazione viene perso (sovrascrittura)
- Ripetere l'esercizio utilizzando invece una stringa contenente più parole, ed inizializzata a piacere

Algoritmo e struttura dati

- Algoritmo
  - Scandire tutta la prima stringa fino al carattere di terminazione ‘\0’
  - Copiare carattere per carattere nella seconda stringa
  - **Aggiungere il carattere di fine stringa!**
- Struttura dati
  - Due variabili stringa ed almeno un indice per scorrere gli array

Programma

```
main()
{
  int i; // volutamente non definito nell'intestazione del for
  char origine [] = "Nel mezzo del cammin di nostra vita";
  char copia [40] ;

  for (i=0; origine[i] != '\0' ; i++)
      copia[i]=origine[i];   /* si esce prima della copia del
                                carattere di fine stringa che,
                                quindi, va aggiunto
                                esplicitamente  */
  copia[i]='\0' ; // FONDAMENTALE !!!
  // da qui in poi si può utilizzare copia come una stringa
  ...
}
```

Stampa di una stringa

Una stringa si può ovviamente stampare anche carattere per carattere

Esempio:

```
int i=0;
char str[]=
     "Nel mezzo del cammin di nostra vita";
…
while (str[i] != '\0') {
     cout<<str[i];
     i++;
}
```

Passaggio alle funzioni

- Per le stringhe valgono la stessa sintassi e semantica del passaggio alle funzioni degli *array*
  - Sono quindi passate sempre per riferimento
  - E' opportuno utilizzare il qualificatore **`const`** per un parametro formale di tipo stringa che non viene modificato dalla funzione

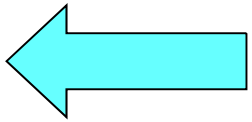
Funzioni di libreria

- Così come per le funzioni matematiche e quelle sui caratteri, il linguaggio C/C++ ha una ricca libreria di funzioni per la gestione delle stringhe, presentata in **`<cstring>`** (**`string.h`** in C)

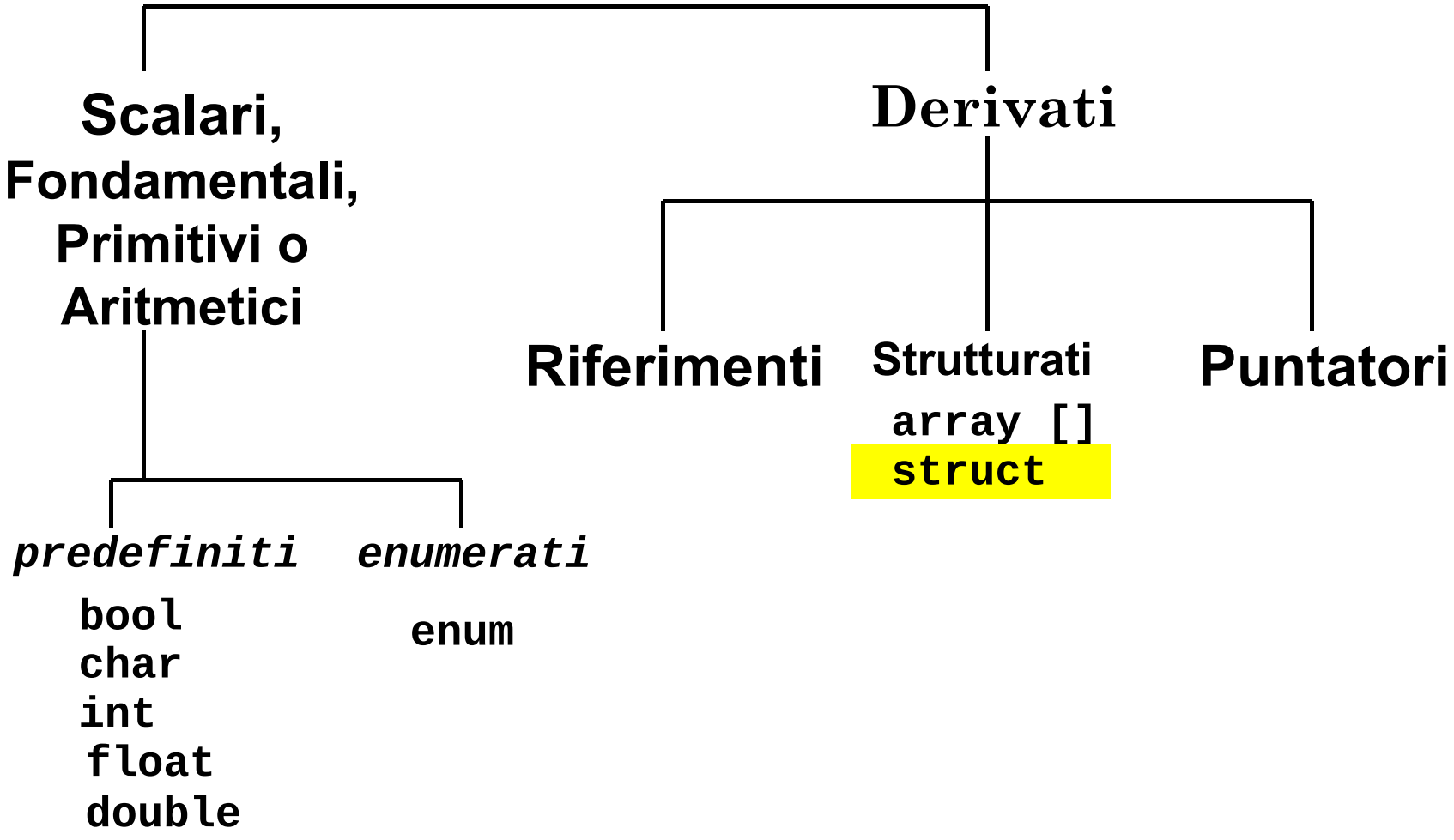
- **`strlen (stringa)`**
  ritorna la lunghezza di una stringa
- **`strcpy(stringa1, stringa2)`**
  copia il contenuto di stringa2 in stringa1 (sovrascrive)
- **`strncpy(stringa1, stringa2, n)`**
  copia i primi n caratteri di stringa2 in stringa1
- **`strcat(stringa1, stringa2)`**
  concatena il contenuto di stringa2 a stringa1
- **`strcmp(stringa1, stringa2)`**
  confronta stringa2 con stringa1: 0 (uguali), >0 (stringa1 è maggiore di stringa 2), <0 (viceversa)

NOTA: In questa contesto minore (o maggiore) indica che stringa1 è alfabeticamente precedente (o successiva) a stringa 2

Contenuto lezione

- Stringhe
- Struct 
  - Operazioni
  - Progettazione strutture dati
  - e passaggio parametri
- Matrici statiche
  - Implementazione
  - Passaggio alle funzioni

Tipo di dato struttura



Problema 1/2

- Dobbiamo scrivere un programma che lavori su delle 'persone'
- In particolare, per ogni persona, nel programma si devono manipolare i seguenti dati:
  - Nome (stringa di al più 15 caratteri)
  - Cognome (stringa di al più 20 caratteri)
  - Luogo di nascita (stringa di al più 20 caratteri)
  - Età (int)
  - Altezza espressa in metri (double)
  - Codice fiscale (stringa di 16 caratteri)

Problema 2/2

- Come facciamo a memorizzare i dati di più persone?
- Se per esempio il programma avesse lavorato **solo** sull'altezza di varie persone, quale tipo di dato avremmo potuto utilizzare per rappresentare tale informazione per tutte le persone?

Risposta

- Un *array* di *double*
  - Col quale implementare magari un vettore dinamico con le tecniche che conosciamo
- Purtroppo però ogni persona è caratterizzata da più di un attributo!
- Come potremmo generalizzare la precedente soluzione continuando ad utilizzare solo i tipi di dato che conosciamo?

Risposta

- Utilizzando un array per ogni attributo, quindi
  - Un *array* di nomi
  - Un *array* di cognomi
  - Un *array* di luoghi di nascita
  - Un *array* di età
  - Un *array* di altezze
  - Un *array* di codici fiscali
- E' però una soluzione pesante e poco leggibile: abbiamo 6 diversi array, mentre quello che vorremmo fare concettualmente è semplicemente rappresentare un solo *array* di **persone**

Soluzione migliore

- Per realizzare una soluzione in cui la struttura dati rappresenti in modo molto più chiaro e semplice i dati del problema, abbiamo bisogno di poter definire direttamente un *tipo di dato persona*
  - Del quale possiamo dire che contiene un nome, un cognome, un luogo di nascita e così via …
- Tutto questo si può fare in C/C++ mediante il costrutto **`struct`**, come mostrato nel seguente esempio

Esempio dichiarazione `struct`

Dichiarazione del nuovo tipo di dato persona

```
struct persona {
      char nome[16];
      char cognome[21];
      char luogo_nascita[21];
      int eta;
      double altezza;
      char codice_fiscale[17];
} ;
```

Utilizzo

- Una volta dichiarato il nuovo tipo di dati persona, è possibile definire variabili di tale tipo

- Ad esempio si può scrivere la seguente definizione (solo in C++, in C va ripetuto **`struct`**, come vedremo in seguito):

```
persona Mario;
```

- Che cos'è la variabile **`Mario`**?
  - Una variabile strutturata composta da tre stringhe, un **`int`**, un **`double`** ed un’altra stringa

Oggetti di tipo struttura

- Oggetto di tipo struttura
  - ennupla ordinata di elementi, detti **membri** o **campi**, ciascuno dei quali ha un suo nome ed un suo tipo
  - Esempio: il campo *nome* nel tipo **`persona`**
- In altri linguaggi il tipo struttura è spesso chiamato **record**
- Un oggetto di tipo struttura differisce da un array per due aspetti:
  - Gli elementi non sono vincolati ad essere tutti dello stesso tipo
  - Ciascun elemento ha un nome

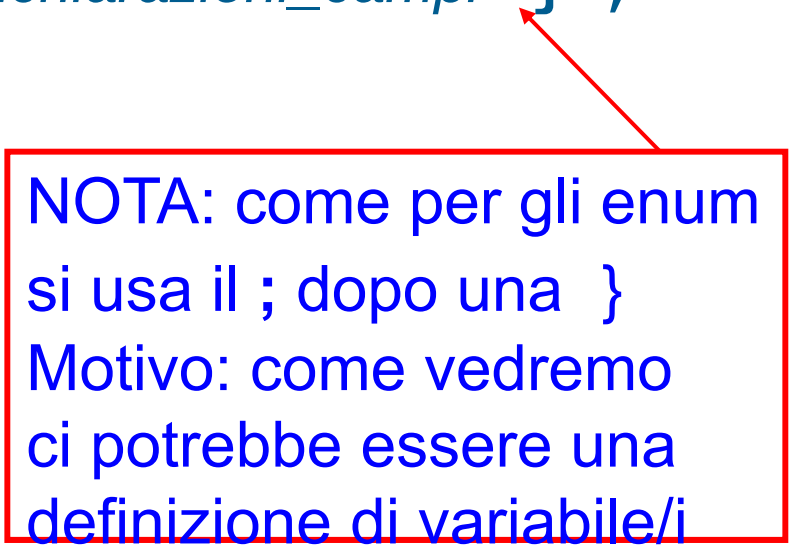
Definizione tipi di dato nuovi

- Mediante il costrutto **`struct`**, si possono di fatto dichiarare nuovi tipi di dato
  - Ad esempio, il precedente tipo **`persona`** è un vero e proprio nuovo tipo di dato, che si può utilizzare a sua volta per definire nuovi oggetti di quel tipo
- Per brevità chiameremo semplicemente *tipi struttura* i tipi di dato dichiarati attraverso il costrutto **`struct`**

Sintassi

- Dichiarazione di un tipo struttura:

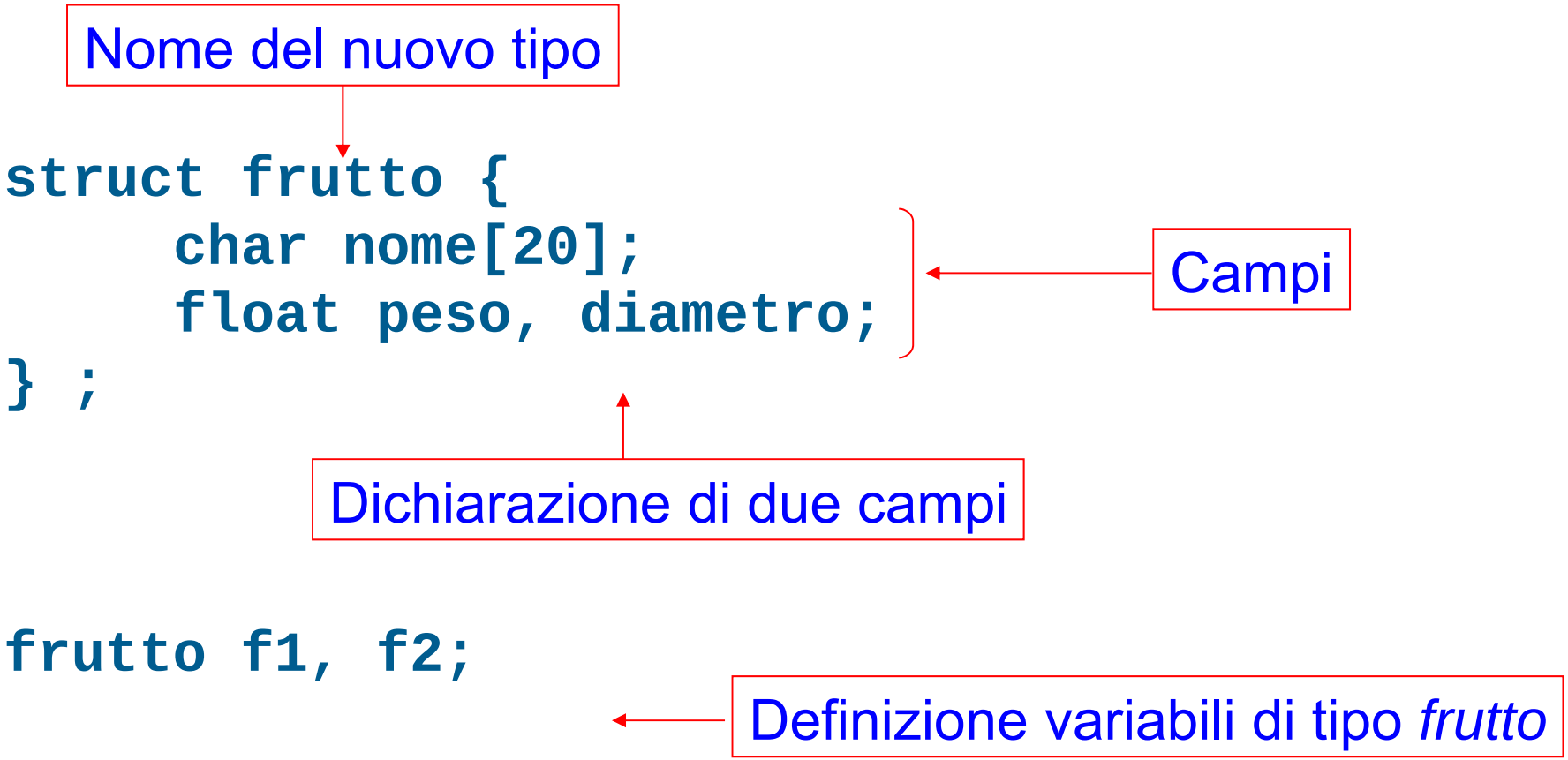
`struct` *<nome_tipo>* **`{`** *<lista_dichiarazioni_campi>* **`}`** **`;`**



- Definizione di oggetti di un tipo strutturato *<nome_tipo>*:

`[const]` *<nome_tipo>*
<identificatore1>, *<identificarore2>*, … **`;`**

Esempio


Nome del nuovo tipo
struct frutto {
char nome[20];
float peso, diametro;
} ;
Campi
Dichiarazione di due campi
frutto f1, f2;
Definizione variabili di tipo *frutto*

Definizione contestuale

- Si possono definire degli oggetti di un dato tipo strutturato anche all'atto della dichiarazione del tipo stesso, con la seguente sintassi

```
[const] struct [<nome_tipo>]
        { <lista_dichiarazioni_campi> }
        <identific_1>, <identific_2>, … ;
```

Nome del nuovo tipo (opzionale)

Esempio:

```
struct frutto {
        char nome[20];
        float peso, diametro;
} f1, f2;
```

Campi

Variabili

Selezione campi

- Per selezionare i campi di un oggetto strutturato si utilizza la *notazione a punto*
- *<nome_oggetto>*`.`*<nome_campo>*

Ad esempio, dato

```
struct frutto {
    char nome[20];
    float peso, diametro; }  f;
```

si può accedere ai campi di f mediante

`f.nome` **`f.peso`** **`f.diametro`**

- che risultano essere normali variabili, rispettivamente di tipo stringa e di tipo float

Esempi:

```
f.peso = 0.34; cout<<f.nome<<endl ;
```

Esempio

```
struct coordinate { int x, y; };

main()
{
    coordinate p1, p2, punto3;
    p1.x=10;  p1.y=20;  p2.x=30;  p2.y=70;
    punto3.x = p1.x + p2.x;
    punto3.y = p1.y + p2.y;
    cout<<"Coordinate risultanti:"
        <<" Ascissa="<<punto3.x<<
        <<" e Ordinata="<<punto3.y<<endl ;
}
```

Esercizio

- Scrivere un programma che chieda all'utente di inserire il nome di un partecipante ad una gara di SCI ed il tempo che ha fatto registrare, espresso in minuti e secondi.

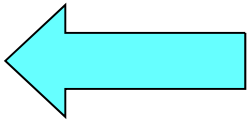
- Il programma deve quindi memorizzare i dati del partecipante in un oggetto struttura che contenga una stringa (per memorizzarvi il nominativo) ed un valore intero, contenente il tempo in secondi. Infine il programma stampa a video i dati appena memorizzati, esprimendo di nuovo il tempo in minuti e secondi.

- Inoltre, il programma stampa a video la lunghezza della stringa contenuta nel campo nominativo.

Esempio:
Inserisci nominativo: Tomba
Tempo di Tomba (minuti e secondi): 2 23
Il vincitore e' Tomba, con il tempo di 2 minuti e 23 secondi
Il nominativo contiene una stringa di 5 caratteri

Contenuto lezione

- Stringhe
- Struct
  - Operazioni
  - Progettazione strutture dati
  - e passaggio parametri
- Matrici statiche
  - Implementazione
  - Passaggio alle funzioni



Inizializzazione

- Un oggetto struttura può essere inizializzato
  - elencando i valori iniziali dei campi fra parentesi graffe
    - Esempio:
    - **`struct coord { int x, y; } ;`**
    - **`coord p1 = {3, 2} ;`**
  - Copiando il contenuto di un altro oggetto dello stesso tipo
    - Esempio:
    - **`coord p2 = p1 ;`**
    - Equivale ad una inizializzazione campo per campo
      - Ossia: **`p2.x = p1.x ; p2.y = p1.y ;`**

Assegnamento

- L'assegnamento tra oggetti di tipo struttura equivale ad una copia campo per campo
  - Esempio:
  - **`coord p1 = {3, 2} ;`**
  - **`coord p2 ;`**
  - **`p2 = p1 ;`**
- I due oggetti devono essere dello stesso tipo struttura
- NON E' CONSENTITO, invece, fare assegnamenti di oggetti struttura **con nomi di tipi diversi**, anche se i due tipi contenessero gli stessi campi

Esempi di assegnamenti

```
struct coordinata { int x; int y;}  p1, p2;
struct coor { int x; int y;}  t1, t2;
int k;
. . .
p2 = p1;
p1 = t2;
t2 = t1;
k = p1;
```

Quali sono validi e quali no?

Risposta

```
struct coordinata { int x; int y;}  p1, p2;
struct coor { int x; int y;}  t1, t2;
int k;
. . .
p2 = p1;
p1 = t2;
t2 = t1;
k = p1;
```

SI
NO
SI
NO

Strutture contenenti array

```
struct abc { int x; int v[5];} ;
abc p1 = {1, {1, 2, 5, 4, 2}} ;
abc p2 ;
p2 = p1 ;
```

A cosa equivale?

Risposta

```
struct abc { int x; int v[5];} ;
abc p1 = {1, {1, 2, 5, 4, 2}} ;
abc p2 ;
p2 = p1 ;
```

Equivale a

```
p2.x = p1.x ;
for (int i = 0 ; i < 5 ; i++)
     p2.v[i] = p1.v[i] ;
```

Domanda

- Si può definire un *array* di oggetti di tipo **struct**?

Domanda

- Esiste quindi un metodo per ottenere la copia tra due *array* senza ricorrere ad un ciclo?

Risposta

- Sì, basta definire un tipo struttura che contiene semplicemente un array
- Se si effettua l'assegnamento tra due oggetti di tale tipo struttura
  - Si ottiene la copia, elemento per elemento,
    - dell'array contenuto nell'oggetto di origine
    - nell'array contenuto nell'oggetto di destinazione

Uso campi o intero oggetto

```
struct frutto { char nome[20]; float peso, diametro; };
main()
{
    frutto f1, f2, f3;
    float somma;
    f1.nome = {‘m’, ‘e’, ‘l’, ‘a’, ‘\0’ }; // ERRATO !!!!!!
    f1.peso=0.26;
    f2.nome={‘a’, ‘r’, ‘a’, ‘n’, ‘c’, ‘i’, ‘a’, ‘\0’};
                                          // ERRATO !!!!!
    f2.peso=0.44;
    somma = f1.peso + f2.peso;
    f3 = f2;
}
```

Utilizzo dei campi

Utilizzo dell'intero oggetto

Domanda

- Si può definire un *array* di oggetti di tipo **struct**?

Risposta

- Ovviamente sì

Esempio:

```
struct coord { int x, y; } ;
coord vett[10] ;
```

- Come si accede agli elementi di un array se tali elementi sono di tipo **struct**?

- E come si accede ai singoli campi di un elemento dell'array nel caso in cui tale elemento sia di tipo **struct**?

Risposta

- Si accede agli elementi con la solita notazione vista finora
- Si accede ai campi di un elemento combinando la notazione per accedere all'elemento con quella per accedere ai campi dell'elemento stesso

Esempio:

```
struct coord { int x, y; } ;
coord vett[10] ;
// Nella prossima riga assegniamo il valore 2
// al campo x del terzo elemento dell'array
vett[2].x = 2;
vett[2].y = 3 ; // Assegniamo 3 al campo y
cout<<vett[2].x<<endl ; // Stampa campo x
```

Domanda

- Quali sono la sintassi e la semantica del passaggio di un array di oggetti di tipo **`struct`**?

Risposta

- Le stesse del passaggio di un array di oggetto di tipo primitivo (quale ad esempio **`int`**)