Tipi di dato strutturati

(Parte 2: stringhe, strutture)

Tipo di dato “stringa”

Stringa

- Oggetto astratto
  - Stringa ➔ *Sequenza di caratteri*

- Letterale stringa (costante senza nome): sequenza di caratteri tra doppi apici
- Esempio:
  `"sono una stringa"`
- Al *cout* abbiamo spesso passato dei letterali di tipo stringa

Stringhe in C/C++

- Nel linguaggio C/C++ non esiste propriamente un tipo stringa
- Le stringhe sono memorizzate come array di caratteri terminati da un carattere **terminatore**
- Tale terminatore è il carattere speciale **'\0'**
  - Numericamente, il suo valore è 0
- Come per i vettori, nella libreria standard del C++ è disponibile anche una implementazione con interfaccia di più alto livello

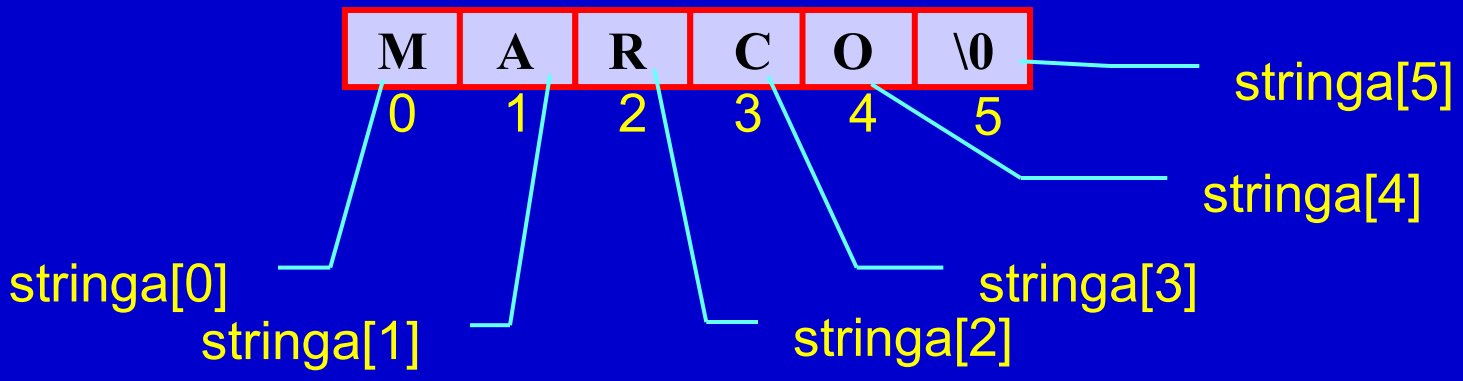
Stringa (sintassi)

SINTASSI della *definizione* di una *variabile* di tipo stringa:

char <identificatore> [<espr-costante>] ;

```
char stringa[6] ;
```

- alloca spazio per 6 oggetti char
- uno va utilizzato per il fine stringa '\0'
- quindi la stringa ha al più 5 caratteri

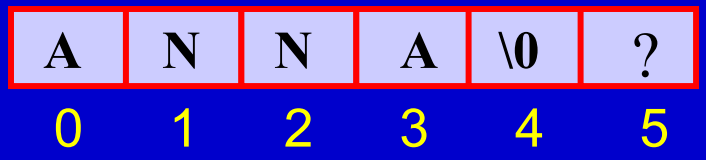


NOTE

La definizione **char stringa[N] ;**

- Alloca spazio per una stringa di al più N-1 caratteri
- E' possibile utilizzarla per stringhe di dimensione inferiore.

Es., **char nome[6] ;**



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

NOTE (2)

- **Una stringa è un array di caratteri**
- **Ma un array di caratteri non è necessariamente una stringa**
- **Affinché un array di caratteri risulti una stringa, è necessario che l'ultimo elemento sia '\0'**

Inizializzazione

Vi sono tre modi per definire ed inizializzare contestualmente una stringa:

```
char nome[6] = { 'M', 'A', 'R', 'C', 'O', '\0' } ;
    /* come un normale array */
char nome[6] = "MARCO" ;
    /* 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 */
```

Dichiarazione e successivo assegnamento

- Se non si tratta di una inizializzazione, l'unico modo per inserire i caratteri di 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 inserito sullo *stdout/stderr*, la stampa dei caratteri termina quando si incontra il terminatore
  - viene utilizzato per riversare il contenuto dello *stdin*, vi finisce dentro la prossima parola (sequenza di caratteri non separati da spazi) Esempio: se sullo *stdin* vi è “ciao mondo”, nella stringa finisce solo “ciao”
- Esistono funzioni simili a *cout* e *cin*, ma che operano su stringa invece che su standard input e standard output
- Esercizio: definire un oggetto di tipo stringa, inizializzarlo, stamparlo, modificarlo da *stdin*, ristamparlo

Errori da evitare con le stringhe

- **Definire un array di caratteri ed inizializzarlo successivamente come una stringa.**

  SEQUENZA DI ISTRUZIONI ERRATA:

```
char nome[6];
nome = “MARCO” ;
```

NO

- **Copiare una stringa in un’altra con l’operazione di assegnamento.**

  SEQUENZA DI ISTRUZIONI ERRATA:

```
char nome[15], cognome[15] ;
nome = cognome;
```

NO

Gli elementi vanno copiati uno alla volta.

Caratteri e stringhe

- E' importante osservare la differenza tra

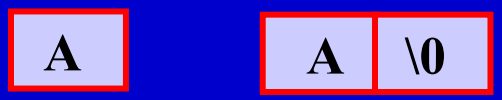
  'A' **carattere A**,

  rappresentabile in una oggetto di tipo **char c='A'**

  "A" **stringa A**,

  rappresentabile in un array, es. **char s[2]="A"**

Tale differenza ha un impatto anche sulla memorizzazione dei relativi dati, rispettivamente:



La stringa è un “vettore particolare”

- I singoli caratteri di una stringa possono anche essere visti come oggetti indipendenti
- Se pensati come stringa sono però parte di un tutt'uno

“MARCO” → ‘M’ ‘A’ ‘R’ ‘C’ ‘O’ '\0'

La stringa è un “vettore particolare” (2)

- Nell’accezione comune, una stringa è una sequenza di caratteri la cui lunghezza può variare
- Di conseguenza, per supportare pienamente il tipo stringa bisognerebbe far riferimento ad un supporto per l’allocazione dinamica della memoria
- Tuttavia, poiché la gestione di dati dinamici sarà trattata in seguito, per ora si prenderà in considerazione solo il caso di
  - **stringhe statiche** (dimensione fissa)
  - **stringhe** con **dimensione massima** definita a tempo di scrittura del programma

Esercizio 1 (*Specifica*)

- **Data una stringa di caratteri, se ne calcoli la lunghezza.**

Esercizio 1 (*Idea*)

- Bisogna scandire tutto il vettore stringa fino al carattere di terminazione `'\0'`, contando i passi che si effettuano

Esercizio 1 (*Algoritmo*)

- Inizializzare una variabile contatore a 0
- Utilizzare un ciclo fino al carattere ‘\0’ ed incrementare la variabile contatore ad ogni passo
- Stampare il valore finale della variabile contatore

Esercizio 9 (*Rappresentazione informazioni*)

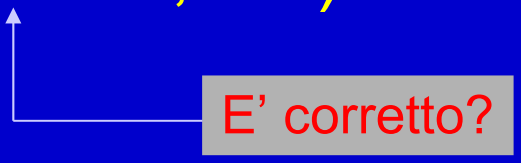
- Serve una variabile stringa
- Servono due variabili ausiliarie (***int***) come contatore (**conta**) e come indice del ciclo (**i**)

Esercizio 1 (*Programma*)

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

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

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

Esercizio 1 (*Programma - vers. 3*)

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

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

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



Esercizio 2 (*Specifica*)

- **Copiare una stringa data in un'altra,**
  - **la stringa di destinazione deve essere memorizzata in un array di dimensioni sufficienti a contenerla**
  - **il precedente contenuto della stringa di destinazione viene perso (sovrascrittura)**

Esercizio 2 (*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

(*Rappresentazione informazioni*)

- Servono due variabili stringa ed almeno un contatore

Esercizio 2 (*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] ; i++)
      copia[i]=origine[i];   /* si esce prima della copia del carattere di
                                fine stringa che, quindi, va aggiunto
 copia[i]=‘\0’ ;              esplicitamente  */
}
```

Esempio

Il programma precedente funziona anche se la stringa iniziale contiene il carattere '0'?

Per esempio:

```
main()
 {
  int i;
  char origine [] = “Abbiamo guadagnato 0 euro”;
  char copia [30] ;

  for (i=0; origine[i]; i++)
      copia[i]=origine[i];
  copia[i]=‘\0’ ;
}
```

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])
    { cout<<str[i];  i++ }
```

Esercizi

- Controllare se una stringa è più lunga di un'altra
- Copiare soltanto i primi 10 caratteri di una stringa in un'altra stringa, inizialmente vuota. [*Att.!: esistono?*]
- Copiare soltanto le vocali di una stringa in un'altra stringa, inizialmente vuota.
- Copiare soltanto le lettere minuscole di una stringa in un'altra stringa, inizialmente vuota. [*Att.!: ricordare la tabella ASCII*]
- Concatenazione (append) di due stringhe: Aggiungere una stringa in fondo ad un'altra stringa, lasciando uno spazio tra le due stringhe [*Att.!: la seconda stringa può essere vuota o no*]
- Verificare se due stringhe sono uguali o diverse
- Data una frase, contare il numero dei caratteri maiuscoli, minuscoli, numerici e dei caratteri non alfanumerici

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)
- 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)

Array di stringhe

Per analogia a quanto detto in precedenza, un array di stringhe si realizza mediante una matrice di tipo char.

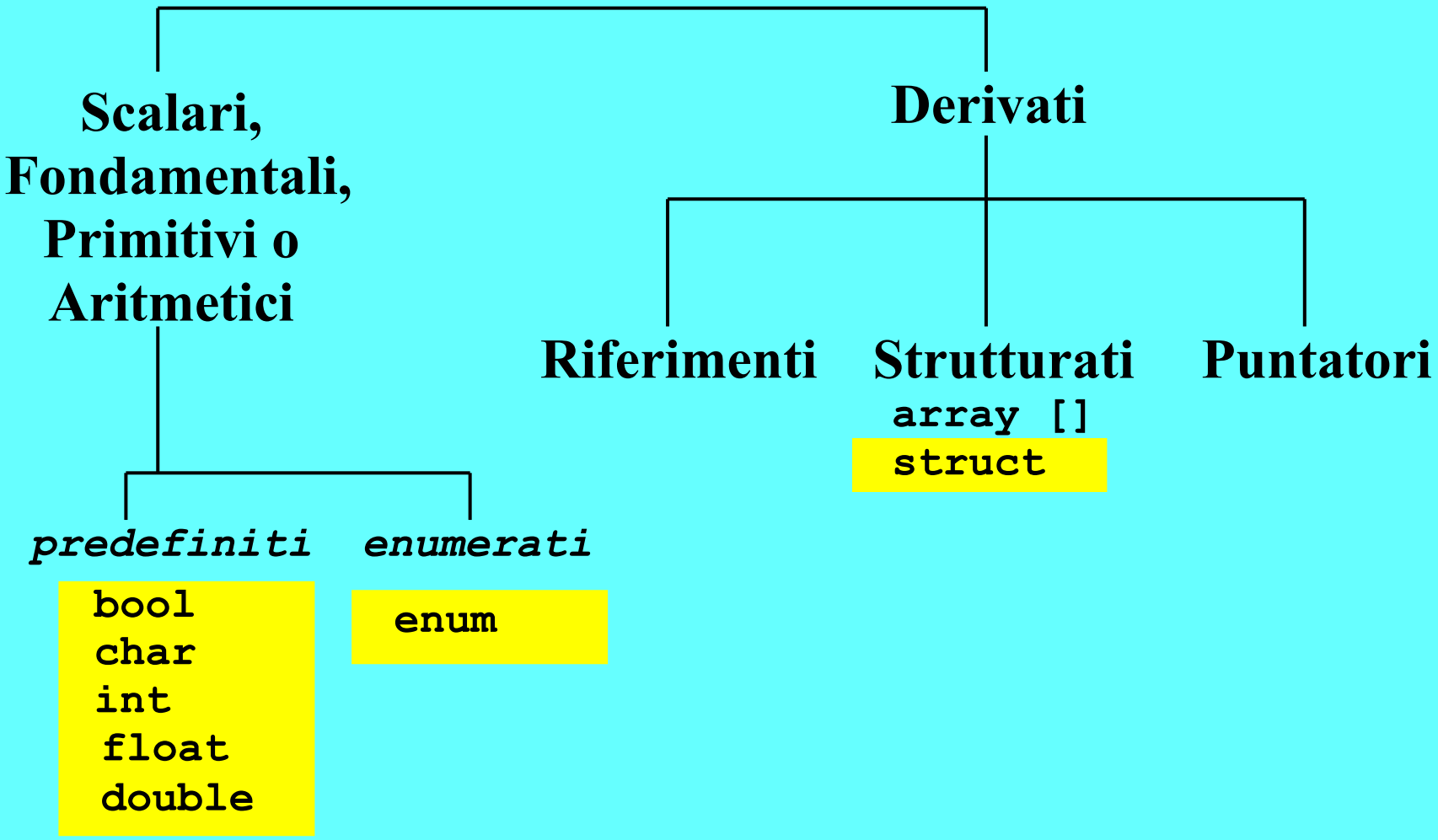
Esempio: **Elenco dei nomi dei giorni della settimana**

char giorni[7][11] = { “lunedi'”, “martedi'”, “mercoledi'”, “giovedi'”, “venerdi'”, “sabato”, “domenica” }

l	u	n	e	d	i	'	\0			
m	a	r	t	e	d	i	'	\0		
m	e	r	c	o	l	e	d	i	'	\0
g	i	o	v	e	d	i	'	\0		
v	e	n	e	r	d	i	'	\0		
s	a	b	a	t	o	\0				
d	o	m	e	n	i	c	a	\0		

Tipo di dato “struttura”

Tipi di dato


Scalari, Fondamentali, Primitivi o Aritmetici
Derivati
Riferimenti
Strutturati
array []
struct
Puntatori
predefiniti
enumerati
bool
char
int
float
double
enum

Esempio

- Caratterizzare alcuni attributi di una “persona”

- **Persona**
  - 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 (**float**)
  - Codice fiscale (**stringa** di 16 caratteri)

Esempio di definizione

Definizione del nuovo tipo di dati **persona**

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

Esempio di utilizzo

- Una volta definito il nuovo tipo di dati **persona**, è possibile definire variabili di questo tipo
- Esempio:

  struct persona Mario, Anna;

- Che cos'è la variabile **Mario**?

→ Una variabile strutturata composta da 3 stringhe, 1 int, 1 float ed un'altra stringa

→ Gli elementi della variabile strutturata sono detti ***membri*** o ***campi***. Ciascun *campo* è caratterizzato da un **tipo** e da un **nome**

Tipo struttura

- Oggetto di tipo struttura
  - *n*-upla ordinata di elementi, detti membri o campi, ciascuno dei quali ha un suo nome ed un suo tipo
- Tipo struttura
  - "Forma" che ha ciascun oggetto di tipo struttura
  - Dichiara il tipo ed il nome di ciascun campo per un generico oggetto di quel tipo struttura
  - Es.: il tipo persona dichiara il nome ed il tipo di tutti (e soli) i campi presenti in un generico oggetto di tipo persona
- In altri linguaggi i tipi struttura sono chiamati **record**

Dichiarazione tipi struttura e definizione variabili

Dichiarazione di un tipo struttura:

struct <nome_tipo> { <lista_dichiarazioni_campi> } ;

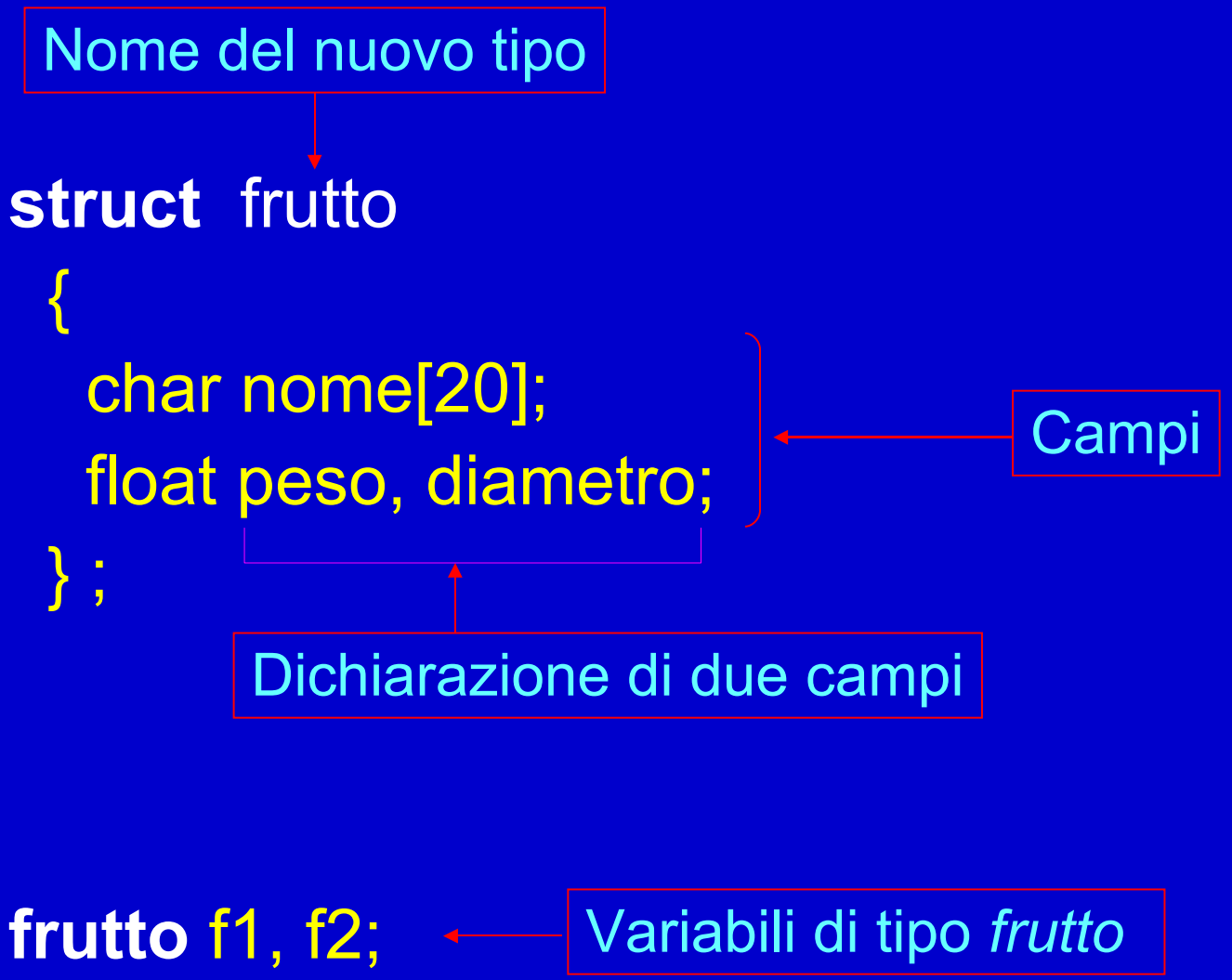
Nome del nuovo tipo

NOTA: Raro caso in cui si usa il ; dopo una }
Motivo: ci potrebbe essere una definizione di variabile/i

Definizione di variabili di un tipo strutturato di nome *nome_tipo*:

[const] <nome_tipo> identificatore1, identificatore2, ... ;

Esempio


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

Definizione di variabili contestuale alla dichiarazione del tipo

Nome tipo: se manca è anonimo

[const] struct [<nome_tipo>]
{ <lista_dichiarazioni_campi> } identif_1, identif_2, ... ;

Esempio

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

Nome nuovo tipo (opzionale)

Campi

Variabili

Selezione dei campi di un oggetto strutturato

- Si usa la “notazione a punto”

- Dato l’esempio precedente

  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

  Es., f.peso = 0.34; cout<<f.nome<<endl ;

Esempio

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


main()
   {
     coordinate p1, p2, punto3;
     p1.x=13;  p1.y=21;  p2.x=32;  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 ;
   }
```

Inizializzazione

- Un oggetto struttura può essere inizializzato

1) elencando i valori iniziali dei campi fra parentesi graffe
   Esempio:
   ```
   struct coord
       { int x, y; } p1 = {3, 2} ;
   ```

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 ;`

Operazioni fra strutture: assegnamento

- Operazioni già viste; inizializzazione e selezione di campo
- Ultima operazione prevista: assegnamento tra oggetti struttura dello stesso tipo
  Esempio:

```
coord p1 = {3, 2} ;
coord p2 ;
p2 = p1 ;
```

  – Equivale ad una copia campo per campo

- **NON E' CONSENTITO, invece, fare assegnamenti di oggetti struttura con nomi di tipi diversi, anche se i tipi avessero gli stessi campi e tipi**

Esempi 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?

Esempi assegnamenti

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

….

`p2 = p1;`	SI
`p1 = t2;`	NO
`t2 = t1;`	SI
`k = p1;`	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 ??

Strutture contenenti array

```
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] ;
```

Utilizzo 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' };
    f1.peso=0.26;
    f2.nome={'a', 'r', 'a', 'n', 'c', 'i', 'a', '\0'};
    f2.peso=0.44;
    somma = f1.peso + f2.peso;
    f3 = f2;
 }
```

Utilizzo dei campi

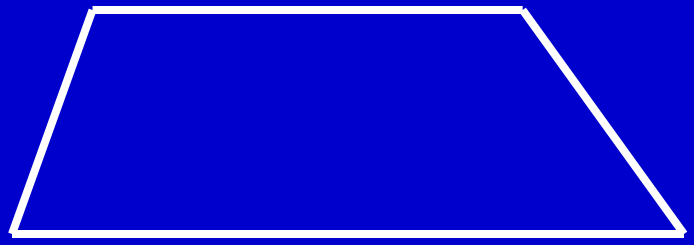
Utilizzo dell'intera variabile struct

Esercizio 3 (*Specifica*)

- **Si determini una struttura dati in grado di rappresentare la figura piana trapezio. Inseriti i dati rappresentativi, si calcoli il perimetro e l'area.**

Esercizio 3 (*Idea-Algoritmo*)

- Un **trapezio** è caratterizzato da:
  - Base maggiore (B)
  - Base minore (b)
  - Lato sinistro (lato_s)
  - Lato destro (lato_d)
  - Altezza (h)



- Per calcolare il **perimetro**, si applica la formula:
  (B+b+lato_s+lato_d)
- Per calcolare l'**area**, si applica la formula:
  ((B+b)*h)/2.

Esercizio 3 (*Rappresentazione informazioni*)

- Come si rappresentano le informazioni relative ad un trapezio?
  - Sono dati omogenei, quindi si potrebbe utilizzare una struttura dati **vettore** (ma bisogna ricordare dove è memorizzato cosa)
  - Pur essendo dati omogenei, si riferiscono ad elementi eterogenei e significativi, per cui si potrebbe utilizzare una struttura dati **struct**

Quale scelta è migliore in assoluto?

Esercizio 3 (*Rappresentazione informazioni*)

- **array**

  vett[0] ~ Base maggiore
  vett[1] ~ Base minore
  vett[2] ~ Lato sinistro
  vett[3] ~ Lato destro
  vett[4] ~ Altezza

- **struct**

```
struct  trapezio
   { float base_maggiore;
     float base_minore;
     float lato_s;
     float lato_d;
     float h;
   } ;
```

Esercizio 3 (*Programma - array*)

```
main()
 {
  float trapezio[5] = {15, 10, 4, 6, 3};
  float perimetro, area;

  perimetro=trapezio[0]+trapezio[1]+trapezio[2]+trapezio[3];
  area=(trapezio[0]+trapezio[1])*trapezio[4])/2.;
  cout<<“Perimetro=”<<perimetro<<” Area=”<<area<<endl ;
}
```

Esercizio 3 (*Programma - struct*)

```
main()
  {
    struct  trapezio
       { float base_maggiore;
          float base_minore;
         float lato_s;
         float lato_d;
         float altezza;
         }  trapezio = {15, 10, 4, 6, 3};
     float perimetro, area;
...
```

Esercizio 3 (*Programma - struct*)

```
...
 perimetro = trapezio.base_maggiore +
                  trapezio.base_minore +
                  trapezio.lato_s + trapezio.lato_d;

  area =

    ( trapezio.base_maggiore + trapezio.base_minore )

    * trapezio.altezza / 2;

  cout<<“Perimetro=”<<perimetro<<” Area=”<<area<<endl ;
}
```

Organizzazione informazioni

- Il precedente esercizio è uno dei casi in cui la struttura fornisce, rispetto all'array:
- Migliore leggibilità
  - I campi sono acceduti mediante nomi significativi
- Migliore organizzazione dei dati
  - Informazioni logicamente correlate stanno nello stesso tipo di dato
- Questa nota è **molto importante** per l'organizzazione dei dati

Passaggio e ritorno oggetti struttura

- Gli oggetti struttura possono essere passati/ritornati per valore
  - Nel parametro formale finisce la copia campo per campo del parametro attuale
- Questo può essere molto oneroso
  - Per esempio se l'oggetto contiene un array molto molto grande
  - Soluzione efficiente ???

Passaggio e ritorno oggetti struttura

- Gli oggetti struttura possono essere passati/ritornati per valore
  - Nel parametro formale finisce la copia campo per campo del parametro attuale
- Questo può essere molto oneroso
  - Per esempio se l'oggetto contiene un array molto molto grande
  - Soluzione efficiente: **passaggio/ritorno per riferimento**
- Però, nel passaggio per riferimento si rischia la modifica indesiderata!
  - Utilizzare il qualificatore **const**

Struttura dati

- Una struttura dati è un
  - insieme di tipi e di oggetti;
  - definito per realizzare un qualche algoritmo.
- Ad esempio, l'insieme di variabili che abbiamo definito come “Rappresentazione delle informazioni” negli esercizi fatti assieme sono appunto delle strutture dati
- Mediante i tipi primitivi, gli array ed il costrutto struct possiamo definire strutture dati di arbitraria complessità
- Ci mancano solo i puntatori per poter definire strutture dati dinamiche