

# **Tipi di dato strutturati**

**(Parte 2: stringhe, strutture)**

# Tipo di dato *stringa*

# Stringa

- Una stringa è una sequenza di caratteri
- Letterale stringa (costante senza nome):  
sequenza di caratteri delimitati da doppi apici
- Esempio:  
`"sono una stringa"`
- All'oggetto *cout* abbiamo spesso passato dei letterali di tipo stringa mediante l'operatore di uscita `<<`

# Stringhe in C/C++

- Nel linguaggio C/C++ non esiste propriamente un tipo stringa
- Il tipo stringa è quindi un oggetto astratto implementato concretamente mediante
  - Un array di caratteri terminati da un carattere **terminatore**
    - Tipicamente 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

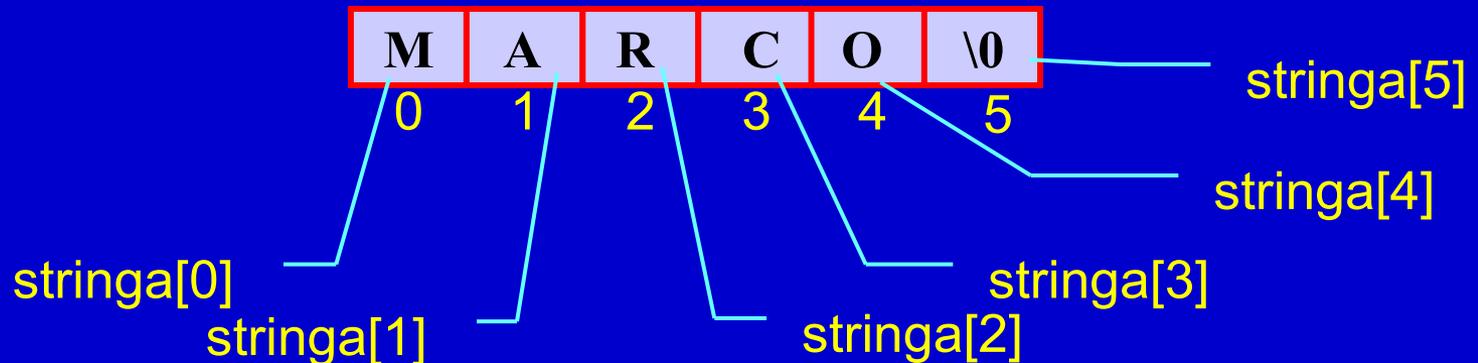
# Stringa (sintassi)

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 1/2

La definizione `char stringa[N]` ;

- Alloca spazio per una stringa di al più N-1 caratteri
- E' possibile utilizzare un array di N caratteri anche per stringhe di dimensione inferiore ad N-1

Es., `char nome[6]` ;

A	N	N	A	\0	?
0	1	2	3	4	5

- 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 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 inizializzare una stringa in una definizione:

```
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 l'array viene dimensionato 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 `<<`, la stampa dei caratteri termina quando 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, scriverci 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 ;
}
```

# 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 un 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:

A

A \0

# 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 l'array che rappresenta la 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
- Serve una variabile ausiliarie (*int*) come indice del ciclo e forse un'ulteriore variabile come contatore del numero di caratteri

# 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++; // poteva bastare la sola variabile i

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

# Stringa diversa

```
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 carattere ' \0 ' è diverso dal carattere ' 0 '

## 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 indice per scorrere gli array

# 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] != '\0' ; i++)
        copia[i]=origine[i]; /* si esce prima della copia del carattere di
                               fine stringa che, quindi, va aggiunto
                               esplicitamente */

    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] != '\0')  
    { cout<<str[i]; i++ }
```

# 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 `stringa2`), <0 (viceversa)

# 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

# Array di stringhe

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

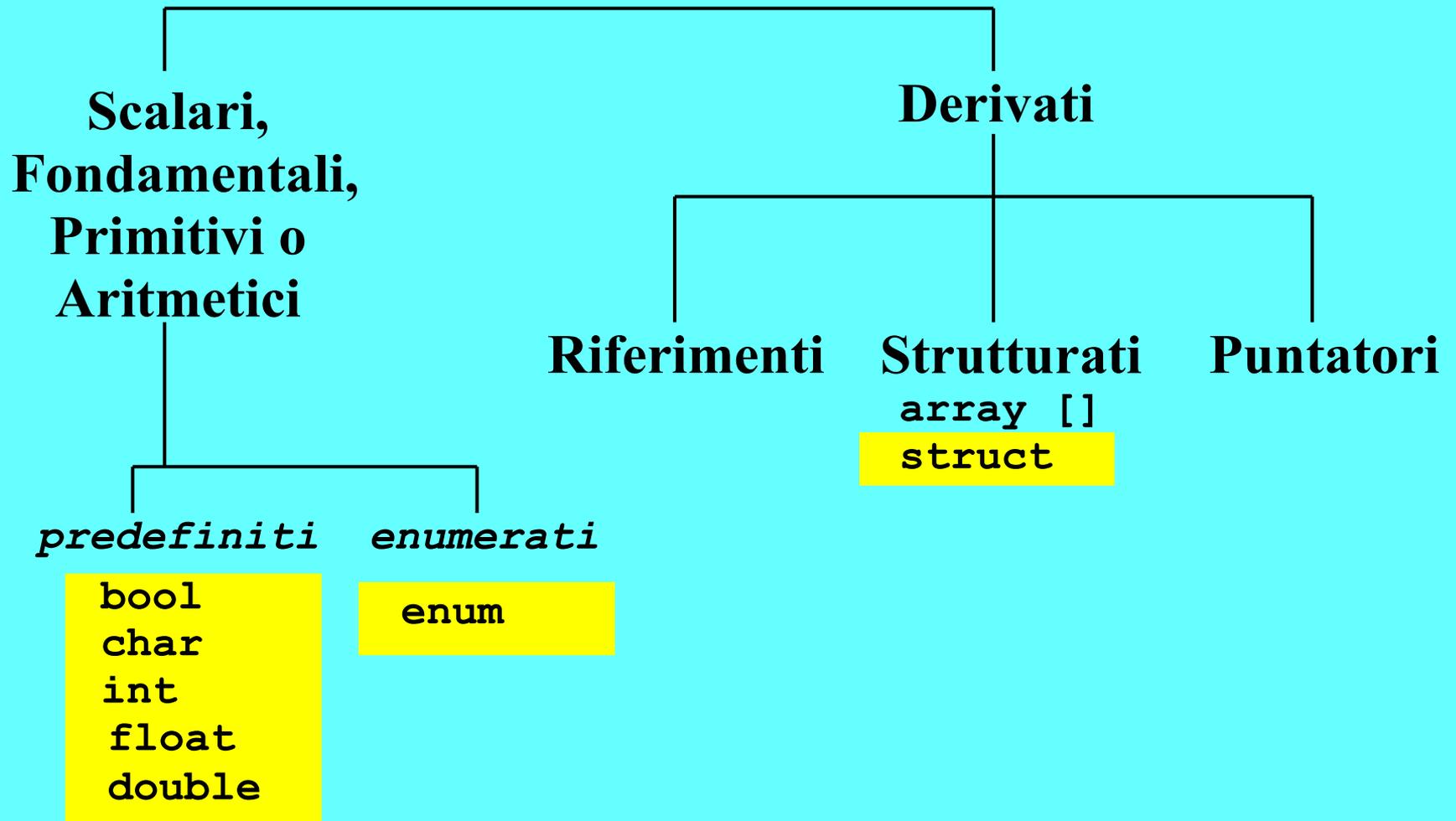
Esempio: **Elenco dei nomi dei giorni della settimana**

```
char giorni[7][11] = { "lunedì", "martedì", "mercoledì",  
"giovedì", "venerdì", "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



# 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 dichiarazione in C/C++

Dichiarazione 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 dichiarato il nuovo tipo di dati **persona**, è possibile definire variabili di questo tipo

- Esempio:

**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**

# Oggetto di 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 di dato basati sul tipo struttura

- Tipo di dato dichiarato mediante il costrutto *struct*  
Es.: il precedente tipo **Persona**
  - Nuovo tipo di dato
  - La dichiarazione del nuovo tipo di dato definisce il tipo ed il nome di ciascun campo di ogni oggetto di quel tipo di dato
  - Es.: il tipo persona dichiara il nome ed il tipo di tutti (e soli) i campi presenti in ogni oggetto di tipo persona
- Per brevità chiameremo semplicemente tipi struttura questi nuovi tipi di dato
- In altri linguaggi i tipi struttura sono spesso 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;

Definizione 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
```

Nome nuovo tipo (opzionale)

```
{
```

```
char nome[20];
```

```
float peso, diametro;
```

Campi

```
} f1, f2;
```

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 ;
```

```
}
```

# Esercizio

- Dalle slide di esercitazione:
  - *struttura.cc*

# 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

- 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 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' }; // 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; ← Utilizzo dei campi
    f3 = f2; ← Utilizzo dell'intera variabile struct
}
CORRETTO
```

# Esercizio

- Dalle slide di esercitazione:
  - *traccia\_classifica\_solo\_elenco.cc*
  - *traccia\_classifica.cc*

## 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 a basso livello (numeri reali), quindi si potrebbe utilizzare un array (ma bisogna ricordare l'indice di ciascuno degli elementi del trapezium)
  - Pur essendo dati omogenei a basso livello (numeri reali), si riferiscono ad elementi concettualmente distinti nel dominio del problema, per cui si potrebbe utilizzare un tipo struttura per poter assegnare un nome distinto a ciascun elemento del trapezio

**Quale scelta è migliore?**

# 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

```
{  
    double base_maggiore;  
    double base_minore;  
    double lato_s;  
    double lato_d;  
    double h;  
};
```

# Esercizio 3 (*Programma - array*)

```
main()
```

```
{  
  double trapezio[5] = {15, 10, 4, 6, 3};  
  double 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
    {
        double base_maggiore;
        double base_minore;
        double lato_s;
        double lato_d;
        double altezza;
    } trapezio = {15, 10, 4, 6, 3};
    double perimetro, area;
    // continua nella prossima slide
```

# 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 ;  
}
```

# Leggibilità ed 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, in base a quanto detto nella lezione sull'ingegneria del codice

# Passaggio e ritorno oggetti struttura 1/2

- 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 2/2

- Passaggio/ritorno per riferimento
- Però, come sappiamo nel passaggio per riferimento si rischia la modifica indesiderata!
  - Utilizzare quindi, come già visto, 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