Tecniche della Programmazione, lez.17

-tipo
- costruttori di tipo
- typedef: alias per i nuovi tipi

-struct (Record), definizione di tipi struct e loro uso in funzioni
- array di structure
- specifica di strutture dati (rappresentazione astratta e rappresentazione concreta)

TIPO (Richiamo)

Un TIPO denota `un insieme di VALORI e OPERAZIONI`

(denota entrambe le seguenti cose ...)

- **la forma della rappresentazione di una certa famiglia di dati**
  - **int** **compl. a 2,**
  - **double** **floating point,**
  - **collezione stringhe** **coppia array+intero**

  in altre parole, come sono costruiti in memoria i VALORI di quel TIPO

- **le OPERAZIONI consentite su quei dati**
  - **per int** **+, -, *, /, ...**
  - **per double** **+, -, *, /, ... (sembrano uguali ma sono diverse ...)**
  - **per la collezione di stringhe, ricerca, aggiunta ...**

COSTRUTTORI DI TIPO

notazioni sintattiche che permettono di definire tipi complessi a partire da tipi elementari

Es. - `int`, `char` **sono tipi elementari**

- `int *` **denota un tipo complesso, puntatore a interi; in questo caso** `'*'` **funge da costruttore di tipo**

- `char []` **denota il tipo array di char**

anche [] funge da costruttore di tipo

NOMI DI TIPO

I nomi di tipi si usano per *definire variabili* ... `int i; char c;` **...**

**L'uso di costruttori di tipo da` luogo a nuovi tipi, con un loro nome usabile per *definire variabili*:** `int *pi`, `char vett[N]`

typedef

typedef e` una parola chiave del C: permette di definire NOMI DI TIPO che sono "*alias*" per altri nomi di tipo

```
typedef TypeName aliasName
```

definisce aliasName come un altro nome per il tipo nome

Es. qui sotto dichiariamo due alias: uno per il tipo `int` ed uno per il tipo `int *`

```
typedef int Intero;
typedef int * PuntIntero;
```

Poi, possiamo definire le variabili `i` e `pi` in modi equivalenti come segue:

```
Intero i;        ≡  int i;
PuntIntero pi;   ≡  int *
```

```
NB  Puntintero pi1, pi2 ≡  int *pi1, *pi2;
```

Analogamente `typedef char Stringa[20]` definisce un alias "`Stringa`" per il tipo "`char[20]`" e poi e` possibile definire una stringa **str** come segue:

```
Stringa str
```

(equivalente a char str[20])

struct e` un costruttore di tipo per i tipi "RECORD"

un record e` una collezione di dati eterogenei (cioe` dati di tipo non necessariamente uguale l'uno all'altro - invece un array ...)

double arr[N]:
- **e` una variabile che rappresenta;**
- **COLLEZIONE DI DATI (elementi)**
  - **con NOME COMUNE arr**
  - **tutti medesimo tipo (double)**
  - **distinti mediate INDICE**

arr

	`arr[0], double`
	`arr[2], double`
	`arr[4], double`
45.88	

un record (una struct) r:
- **e` una variabile che rappresenta;**
- **COLLEZIONE DI DATI (*campi*)**
  - **con NOME COMUNE r**
  - **ev. di TIPI DIVERSI**
  - **distinti mediante NOMECAMPO**

r

7.3	ascissa	`double`
4.2	ordinata	`double`
"rosso"	colore	`char[11]`

definizione di una variabile di tipo record/struct

due passi:

(1) definizione del TIPO di struttura, mediante costruttore `struct`

(2) definizione (dichiarazione) della variabile di quel tipo

Es. (1)

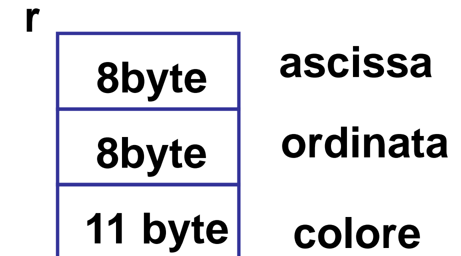
```
struct tipoPunto {
        double ascissa;
        double ordinata;
        char colore[11];
};
```

il TIPO si chiama
`struct tipoPunto`
E puo` essere usato per definire variabili

(2)

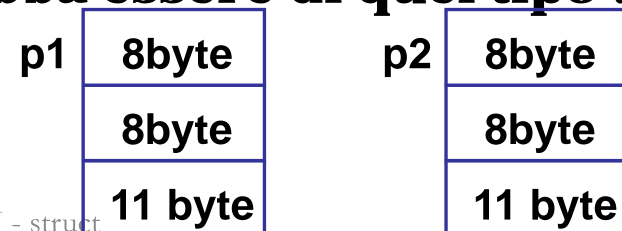
```
struct tipoPunto r;
```

```
/* allocazione statica di una locazione,
con identificatore r, capace di contenere
2 locazioni double e un array di 11 char */
```



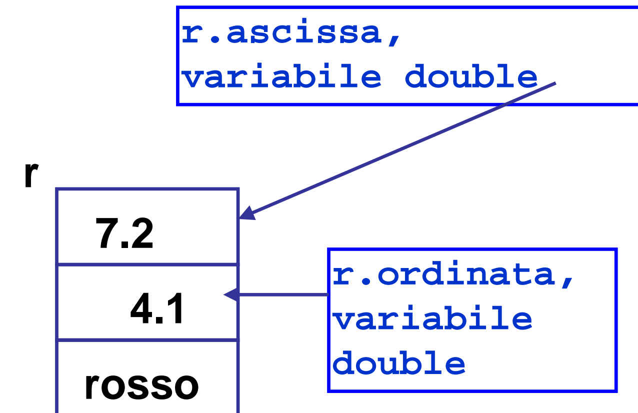
dato che `struct tipoPunto` ora e` un nome di tipo, lo si puo` usare per definire qualunque variabile che debba essere di quel tipo ...

```
struct tipoPunto p1, p2;
```



USO DI UNA STRUTTURA

ACCESSO / MEMORIZZAZIONE, mediante *dot.notation*



- **campo ascissa della struct r = 7.2:**

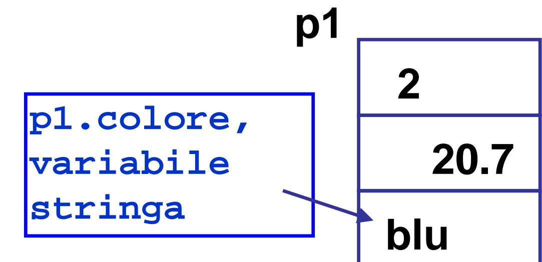
```
r.ascissa = 7.2
```

- **campo ordinata ... = 4.1**

```
r.ordinata = 4.1
```

- **campo ordinata di p1 = 20.7**

```
p1.ordinata = 20.7
```



- **stampa ascissa di p1**

```
printf(" ... %g ...", p1.ascissa);
```

- **lettura da input del campo "ordinata" del record r**

```
scanf("%lf", &r.ordinata);
```

- **lettura colore della struct (o record) p1 da input**

```
scanf("%s", p1.colore);
```

remember: e` una "variabile array"

- **stampa del punto p1**

```
printf("(%g, %g):%s", p1.ascissa, p1.ordinata, p1.colore);
```

USO DI UNA STRUTTURA

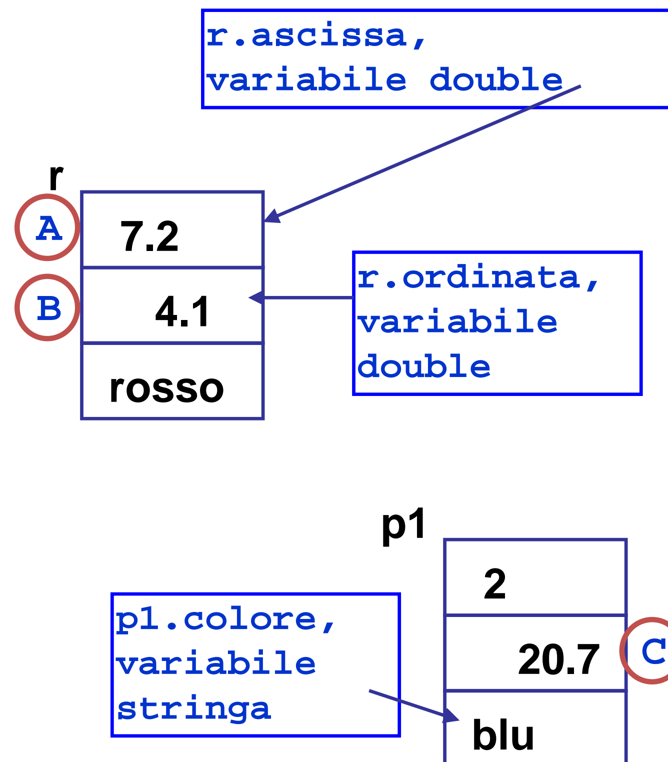
ACCESSO / MEMORIZZAZIONE, mediante ***dot.notation***

- **campo ascissa della struct r = 7.2:**

```
r.ascissa = 7.2
```
(A)

- **campo ordinata ... = 4.1**

(B)
```
r.ordinata = 4.1
```



- **campo ordinata di p1 = 20.7**

(C)
```
p1.ordinata = 20.7
```

- **stampa ascissa di p1** (stampa 2)

```
printf(" ... %g ...", p1.ascissa);
```

- **lettura da input del campo "ordinata" del record r**

(legge il 4.1)
```
scanf("%lf", &r.ordinata);
```

- **lettura colore della struct (o record) p1 da input**

(legge il blu)
```
scanf("%s", p1.colore);
```

- **stampa del punto p1**

```
printf("(%g, %g):%s", p1.ascissa, p1.ordinata, p1.colore);
```

remember: e` una "variabile array"

(2, 20.7: blu

primo uso di struct, punto colorato medio - 1/3 -

esercizio leggere due punti colorati e calcolare&stampare il punto medio (il colore del punto medio e` nero se i due punti hanno colore diverso, senno` e` il colore comune)

punto colorato? **una variabile "punto colorato" e` un insieme di dati comprendenti una ascissa, una ordinata (numeri reali) e un colore (stringa di al + 10 caratteri)**

```
struct punto {
   double ascissa;
   double ordinata;
   char colore[11];
};

/* i due punti colorati */
struct punto p1, p2;

/* successivamente riempiti */


/* il punto medio */
struct punto pMedio;
```

p1

2
2
blu

p2

6
4
rosso

pMedio

-
-

Algoritmo? ☺

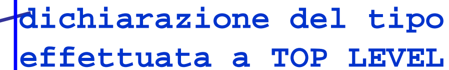
primo uso di struct, punto colorato medio - 2/3 -

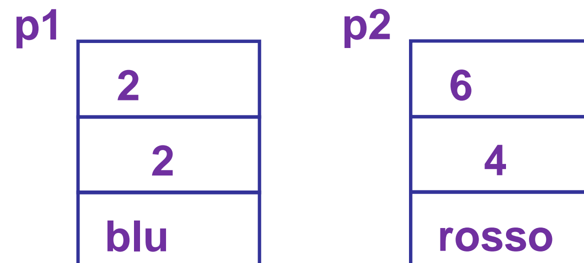
```
#include <stdio.h>
   struct punto {
      double ascissa;
      double ordinata;
```

```
Algoritmo …

0) ci servono i tre punti … p1, p2, pMedio

1) lettura dati (chiedere e leggere p1 e p2)

2) calcolo pMedio
   1)pMedio.ascissa, con la geometria
   2)pMedio.ordinata …
   3)pMedio.colore: applichiamo il criterio "se
     uguali colori … senno` nero)

3) stampa pMedio … (ascissa, ordinata, colore)
```

primo uso di struct, punto colorato medio - 3/3 -

```
#include <stdio.h>
   struct punto {
      double ascissa;
      double ordinata;
      char colore[11]
};

int main() {        struct punto p1, p2, pMedio;      /* 0 */

  printf("ascissa primo: ");                          /* 1 */
  scanf("%lf", &p1.ascissa);
  printf("ordinata primo: ");
  scanf("%lf", &p1.ordinata);
  printf("colore primo: ");
  scanf("%s", p1.colore);

  printf("ascissa secondo: ");
  scanf("%lf", &p2.ascissa);
  printf("ordinata secondo: ");
  scanf("%lf", &p2.ordinata);
  printf("colore secondo: ");
  scanf("%s", p2.colore);
...
```





primo uso di struct, punto colorato medio - 4/3 -!

```
...
  pMedio.ascissa = (p1.ascissa + p2.ascissa)/2;       /* 3.1 */
  pMedio.ordinata = (p1.ordinata + p2.ordinata)/2;    /* 3.2 */
  if (strcmp(p1.colore, p2.colore)!=0)                /* … */
    strcpy(pMedio.colore, "nero");
  else
    strcpy(pMedio.colore, p1.colore);

  printf(punto medio: (%g, %g, %s)\n", pMedio.ascissa,
     pMedio.ordinata, pMedio.colore);                 /* … */

printf("\nFINE PROGRAMMA\n\n");
return 0;
}
```

p1	p2
2	6
2	4
blu	rosso

pMedio
4
3
nero

```
#include <stdio.h>
   struct punto {
      double ascissa;
      double ordinata;
      char colore[11]
};
int main() {
         struct punto p1, p2, pMedio;
...
```

NB - perche' abbiamo definito struct punto { ...} TOP LEVEL e non nella main()???

perche' senno` nessuno al di fuori della main() potrebbe "vedere" e usare quella definizione (ad es. altre funzioni non potrebbero ...)

varie sulle struct ...

inizializzazione di una struct in definizione

```
struct punto {  ...
};
                        struct punto p1 = {6.7,6.6,"viola"};
```

definizione alternativa di 2 variabili struct

```
struct punto {
   double ascissa, ordinata;
   char colore[11];
} p1, p2;
/* possiamo definirne altre dopo …
 es. struct punto p4, p5, p6    */
```

definizione alternativa di 2 var struct, con TIPO ANONIMO

```
struct {
   double ascissa, ordinata;
   char colore[11];
} p1, p2;
/* dopo, non possiamo definirne altre */
```

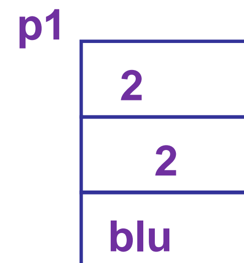
definizione di tipi struct con typedef (nuovo NOME DI TIPO)

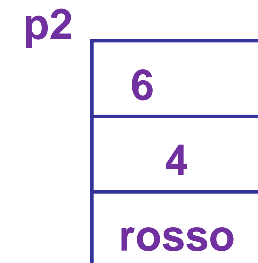
```
typedef
       struct punto
       TipoPunto;

/* e poi possiamo definirci var… */

TipoPunto  p1, p2;
```

```
typedef            /* con tipo anonimo */
    struct {
      double ascissa, ordinata;
      char colore[11];
    }
     TipoPunto;
/*e poi possiamo definirci var… */
TipoPunto p1, p2;
```

potremmo riscrivere quel programma usando il tipo TipoPunto, invece di struct punto …

```
#include <stdio.h>
   struct punto {
      double ascissa;
      double ordinata;
      char colore[11]
};

typedef struct punto
        TipoPunto;           /* NB primo carattere maiuscolo, per indicare
                             che e` un nome di tipo */

int main() {
    TipoPunto p1, p2, pMedio;

  printf("ascissa primo: ");
  scanf("%lf", &p1.ascissa);
  printf("ordinata primo: ");
  scanf("%lf", &p1.ordinata);
...
```



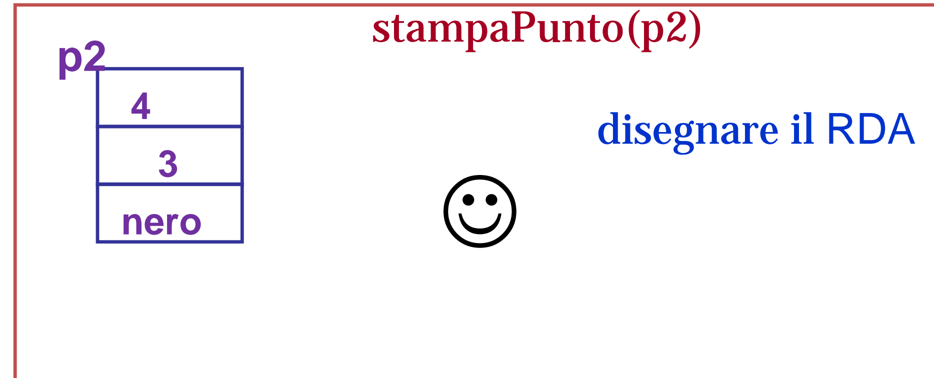


Funzioni che usano struct

Funzioni che ricevono struct

NB passaggio per valore ...

```
void stampaPunto( struct punto p) {
  printf("(%g, %g, %s)\n", p.ascissa, p.ordinata, p.colore);
return;
}
```

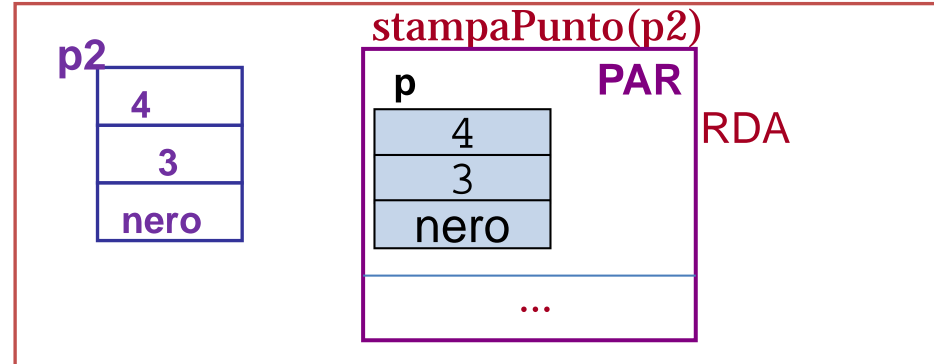


Funzioni che usano struct

NB passaggio per valore ...

Funzioni che ricevono struct

```
void stampaPunto( struct punto p) {
  printf("(%g, %g, %s)\n", p.ascissa, p.ordinata, p.colore);
return;
}
```



Funzioni che restituiscono struct

```
TipoPunto puntoMedio (TipoPunto primo, TipoPunto secondo) {
   TipoPunto pM;

  pM.ascissa = ...
  pM.ordinata = ...
  if (...)
    strcpy(pM.colore, ...);
  else strcpy(pM.colore, ...);
return pM;
}
```

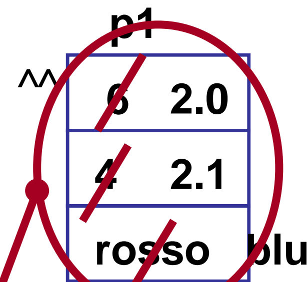
NB p1, p2 per valore; durante la restituzione del risultato, i campi di pM vengono passati fuori per essere copiati, membro a membro, in quelli di pMedio

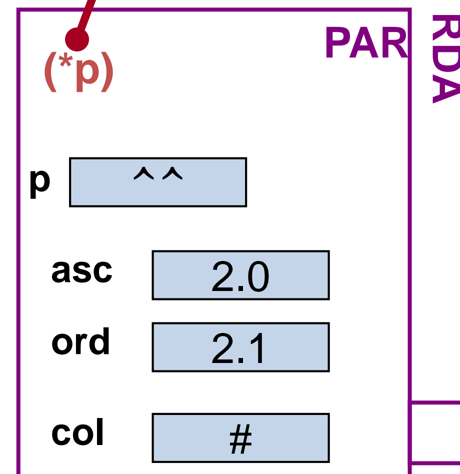
```
chiamata (p1, p2, pMedio sono punti nella funzione chiamante)
pMedio = puntoMedio(p1, p2);
```

Funzioni con side effect su struct - 1/3 -

Funzioni che provocano side effect su struct: definiamo una funzione che riceve un punto e tre valori (ascissa, ordinata e colore) e assegna i tre valori al punto.
Una specie di operatore di assegnazione ...

```
void assegnaPunto ( struct punto *p,
       double asc, double ord, char *col) {
...
return;
}

int main() {
  struct punto p1={...};

  ...
  assegnaPunto(&p1, 2.0, 2.1, "blu");
  ...
return 0;
}
```






```
modifiche sulla locazione *p = modifiche sulla struct p1:
i campi sono  (*p).ascissa,  (*p).ordinata,  (*p).colore
```

```
NB        L'ESPRESSIONE (*p) viene usata nell'ambiente della funzione, per
denotare quello che fuori della funzione (ambiente della funzione chiamante)
e` indicato con p1

modifiche sulla locazione *p   ==   modifiche su p1
```

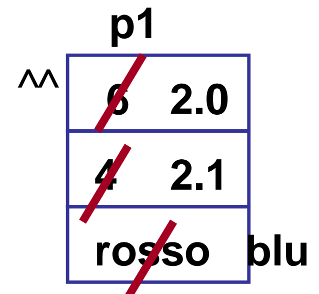
come e` definite assegnaPunto? ☺

Funzioni con side effect su struct - 2/3 -

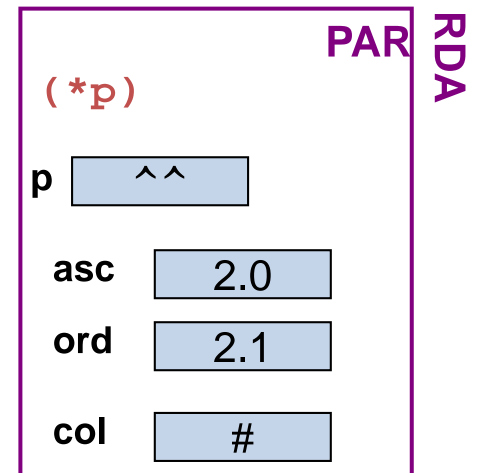
Funzioni che provocano side effect su struct

```
void assegnaPunto ( struct punto *p,
                    double asc, double ord, char *col) {
    (*p).ascissa = asc;
    (*p).ordinata = ord;
    strcpy((*p).colore, col);

return;
}
```

```
modifiche sulla locazione *p = modifiche sulla struct p1:

I campi si chiamano (*p).asc, (*p).ord, (*p).colore
```



assegnaPunto(&p1, 2.0,2.1,"blu");

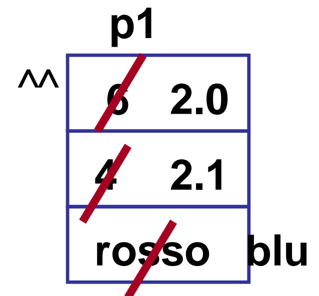


```
void assegnaPunto ( struct punto *p,
                    double asc, double ord, char *col) {
    (*p).ascissa = asc;
    (*p).ordinata = ord;
    strcpy((*p).colore, col);

return;
}
```

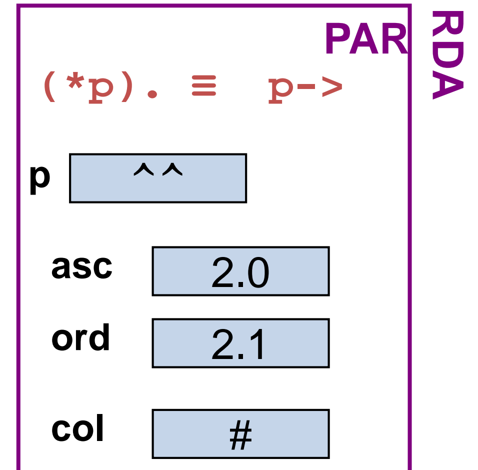
modifiche sulla locazione *p = modifiche sulla struct p1:

I campi si chiamano (*p).asc, (*p).ord, (*p).colore

o anche p->ascissa, p->ordinata, p->colore



assegnaPunto(&p1, 2.0,2.1,"blu");



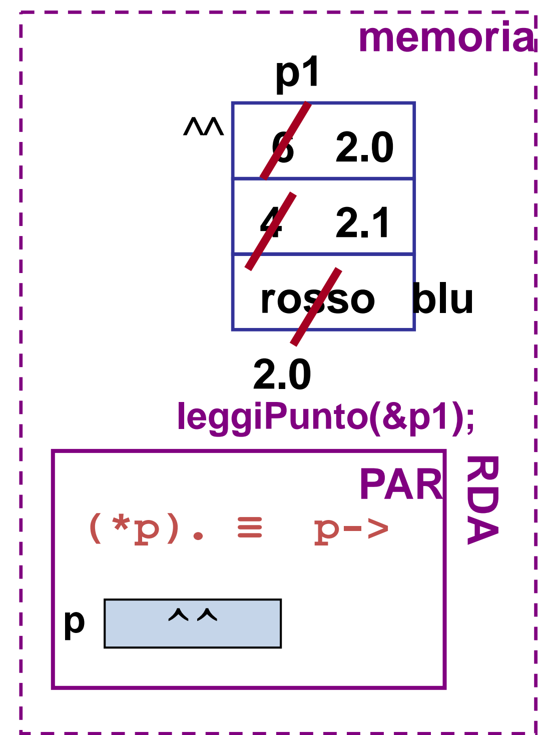
NOTAZIONE

```
per      struct punto * p;
vale     (*p).ascissa ≡ p->ascissa
quindi possiamo scrivere
                    p->ascissa = asc;
                    p->ordinata = ord;
                    strcpy(p->colore, col);
```

prima di proseguire definire leggiPunto(), che riceve un punto e ne legge i dati da input ☺

Funzioni con side effect su struct - 3/3 -

usiamo la notazione freccia ...
e anche typedef, per definiire l'alias TipoPuntoColorato




```
typedef struct punto
        TipoPuntoColorato;

void leggiPunto (TipoPuntoColorato *p) {
   printf("ascissa? ");
   scanf("%lf", &(p->ascissa));
   printf("ordinata? ");
   scanf("%lf", &(p->ordinata));
   printf("colore? ");
   scanf("%s", p->colore);
return;
}
```

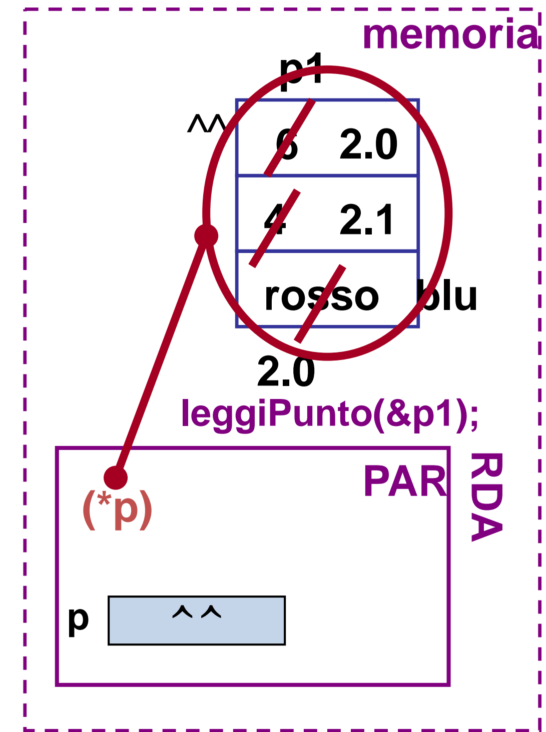
```
Input: 2.0 2.1 blu e poi return
```

equivalente a

```
void leggiPunto (TipoPuntoColorato *p) {
   printf("ascissa? ");      scanf("%lf", &((*p).ascissa));
   printf("ordinata? ");     scanf("%lf", &((*p).ordinata));
   printf("colore? ");       scanf("%s", (*p).colore);
return;
}
```

Funzioni con side effect su struct - 3/3 - note

```
typedef struct punto
        TipoPuntoColorato;

void leggiPunto (TipoPuntoColorato *p) {
    printf("ascissa? ");
    scanf("%lf", &(p->ascissa));
    printf("ordinata? ");
    scanf("%lf", &(p->ordinata));
    printf("colore? ");
    scanf("%s", p->colore);
return;
}
```



`Input: 2.0 2.1 blu e poi return`

NB

TipoPunto, TipoPuntoColorato, struct punto sono tipi EQUIVALENTI (posso usare valori di uno di questi tipi ovunque ci si aspettino valori degli altri tipi … i valori sono fatti nella stessa maniera …)

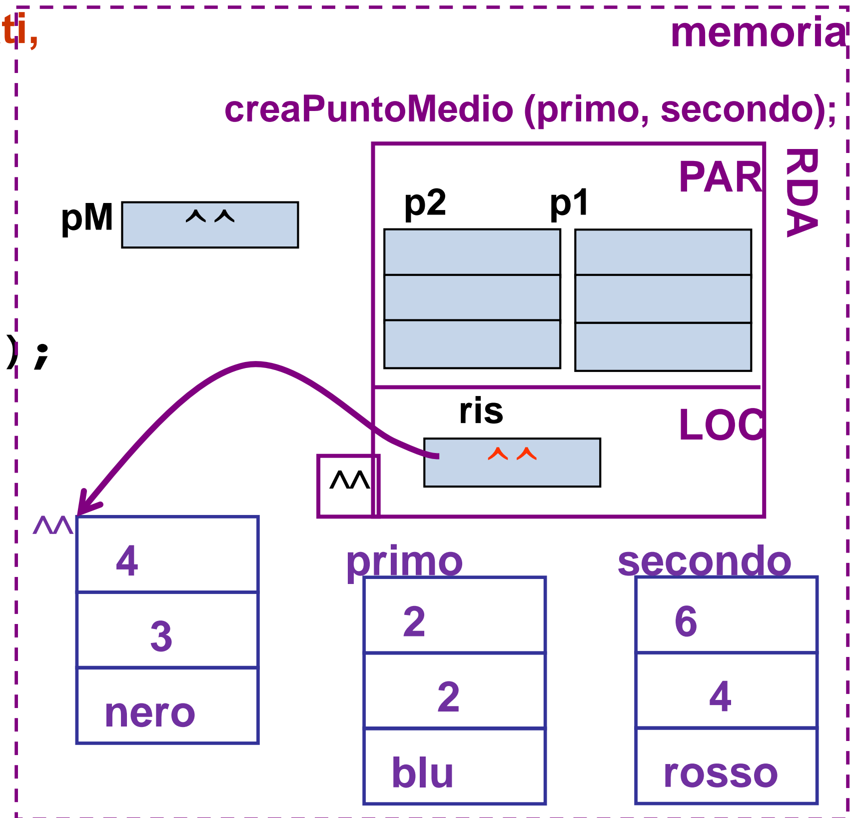
IDEM per

TipoPunto * ≡ TipoPuntoColorato * ≡ struct punto *

Funzioni che restituiscono puntatori a struct - 1/5 -

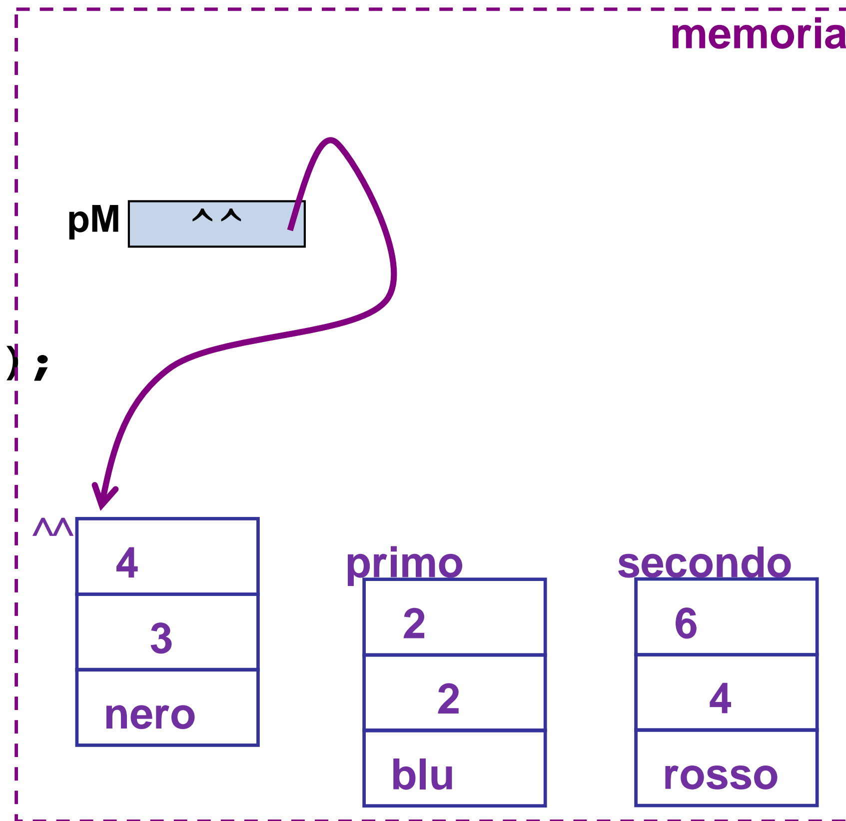
facciamo una funzione che riceve due punti colorati, "crea" il punto medio, e ne restituisce il puntatore

```
int main() {
   TipoPunto primo, secondo, *pM;
   ...
 pM = creaPuntoMedio(primo, secondo);
 ...
 stampaPunto(?);

return 0;
}
```



Funzioni che restituiscono un puntatore a struct

```
/* "creazione" punto medio */
int main() {
  TipoPunto primo, secondo, *pM;
  ...
 pM = creaPuntoMedio(primo, secondo);
 ...
 stampaPunto(?);

return 0;
}
```

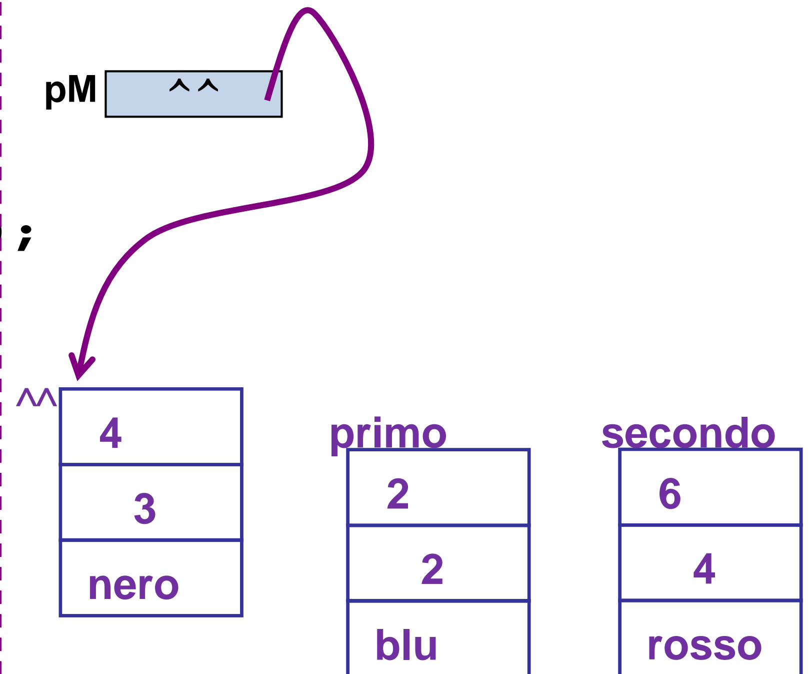


Parametro attuale di stampaPunto ?

Intestazione della definizione di creaPuntoMedio ?

Funzioni che restituiscono un puntatore a struct

```
/* "creazione" punto medio */
int main() {
  TipoPunto primo, secondo, *pM;
  ...
 pM = creaPuntoMedio(primo, secondo);
 ...
 stampaPunto(*pM);

return 0;
}
```



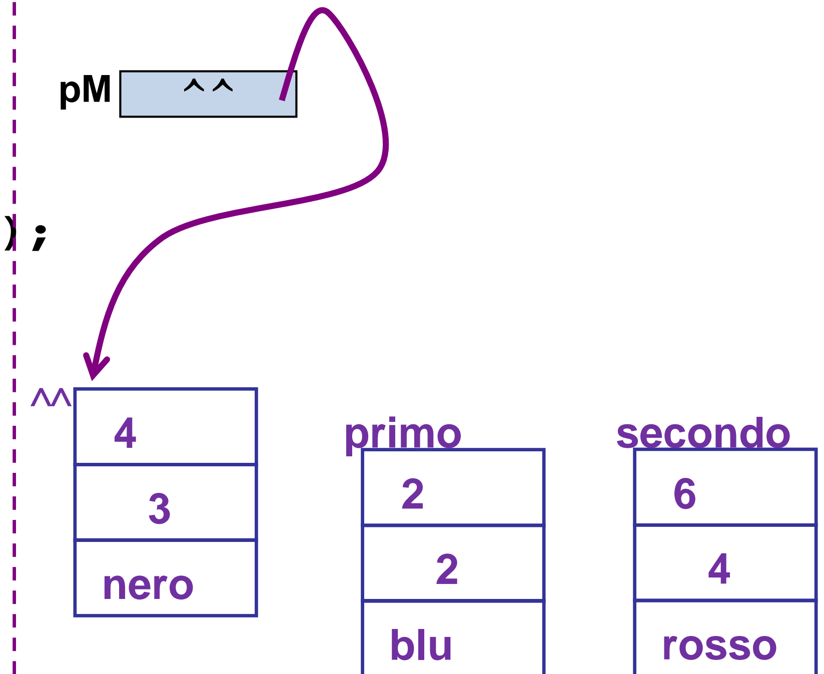
NB

```
pM e` il puntatore alla struct;

*pM e` (il contenuto del)la locazione
puntata da pM; cioe` la   struct;
e stampaPunto si aspetta una struct
(non un puntatore a struct ...)
```

Intestazione della definizione di creaPuntoMedio ?

Funzioni che restituiscono un puntatore a struct

```
/* "creazione" punto medio */
int main() {
  TipoPunto primo, secondo, *pM;
  ...
 pM = creaPuntoMedio(primo, secondo);
 ...
 stampaPunto(*pM);

return 0;
}
```



NB

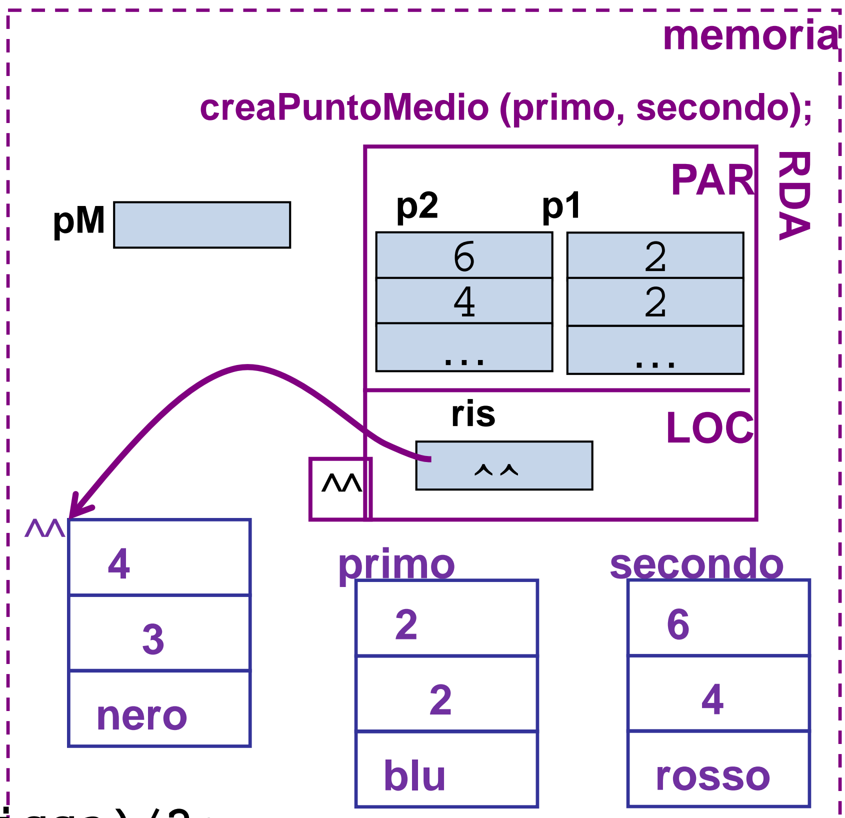
`pM e` il puntatore alla struct;`

`*pM e` (il contenuto del)la locazione puntata da pM; cioe` la struct; e stampaPunto si aspetta una struct (non un puntatore a struct ...)`

NB esempio di intestazione per la funzione

```
TipoPunto * creaPuntoMedio (TipoPunto p1, TipoPunto p2)
```

Funzioni che restituiscono un puntatore a struct



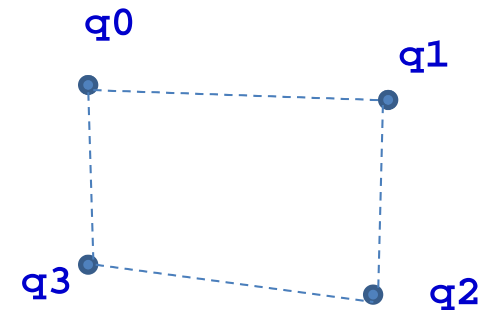

```
TipoPunto * creaPuntoMedio(
                 TipoPunto p1,
                 TipoPunto p2) {

  TipoPunto *ris;

 ris = malloc(sizeof(TipoPunto));

 if (!ris)
   printf("messaggio di errore");
 else {

   ris->ascissa = (p1.ascissa+p2.ascissa)/2;
   ris->ordinata = ...;
   if (strcmp(p1.colore, p2.colore))
     strcpy(ris->colore, "nero");
   else strcpy(ris->colore, p2.colore);
 }
return ris;
}
```

Quadrilatero

Dati i vertici di un quadrilatero q1, q2, q3, q4 (punti colorati), definire un tipo opportuno per allocare dinamicamente il quadrilatero e poi determinare se il quadrilatero e` equilatero e/o isotetico (lati || assi)

SCHEMA DI PROGRAMMA

- algoritmo,
- strutture dati coinvolte,
- main(), con chiamate di funzioni di servizio (es. sottoprogrammi per sottoproblemi)
- definizioni delle funzioni



NB usiamo TipoPunto invece che TipoPuntoColorato, per problemi di spazio

Quadrilatero

Dati i vertici di un quadrilatero q1, q2, q3, q4 (punti colorati), definire un tipo opportuno per allocare dinamicamente il quadrilatero e poi determinare se il quadrilatero e` equilatero e/o isotetico (lati || assi)

algoritmo, strutture dati coinvolte,

0) variabile quad et al. `variabile quad`

1) alloc. din. quadrilatero = array 4 punti
se problemi, … fine

2) lettura vertici e loro stampa (controllo)

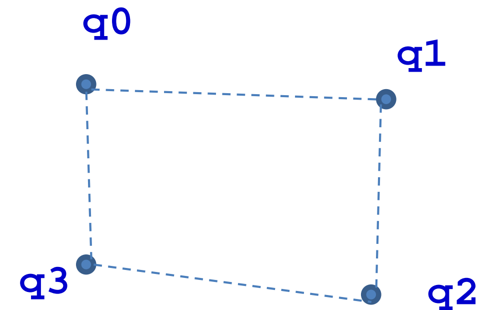
3) `equi` = «quad e` equilatero» = `equilatero(quad)`

4) `iso` = «quad e` isotetico» = `isotetico(quad)`

5) se (`equi==TRUE`) … messaggio

6) se (`iso==TRUE`) … messaggio

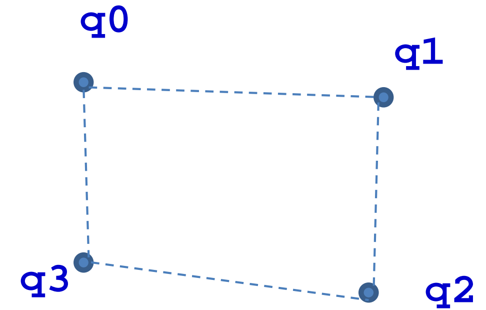
7) se entrambi … quadrato!



NB usiamo TipoPunto invece che TipoPuntoColorato, per problemi di spazio

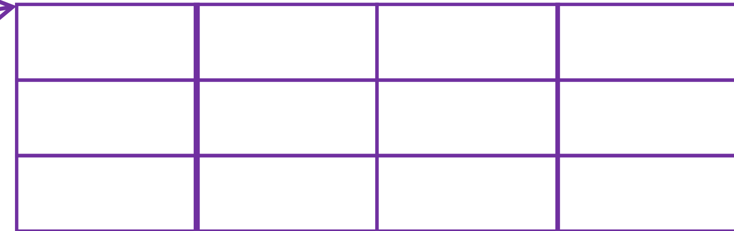
Quadrilatero - la struttura dati

```
struct punto {
    double ascissa, ordinata;
    char colore[11];
};
typedef struct punto TipoPunto

int main() {
```



~~TipoPunto quad ?~~

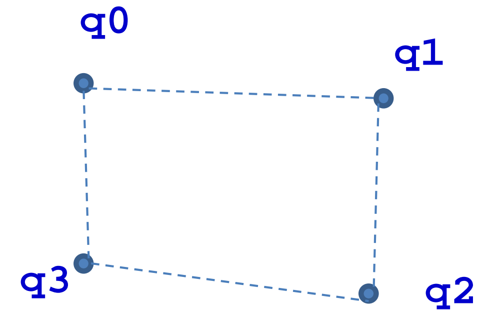
```
 TipoPunto * quad
```



```
 int iso, equi, i;

 quad = malloc (4 * sizeof(TipoPunto));
```

Quadrilatero - la struttura dati

```
struct punto {
   double ascissa, ordinata;
   char colore[11];
};
typedef struct punto TipoPunto

int main() {

 TipoPunto * quad
 int iso, equi, i;

 quad = malloc (4 * sizeof(TipoPunto));
 if (quad==NULL) {
    printf("... AARGH ...");
    return 0;
 }
 leggiQuadrilatero(quad);
 stampaQuadrilatero(quad);
 equi = equilatero(quad);
 iso = isotetico(quad);
```

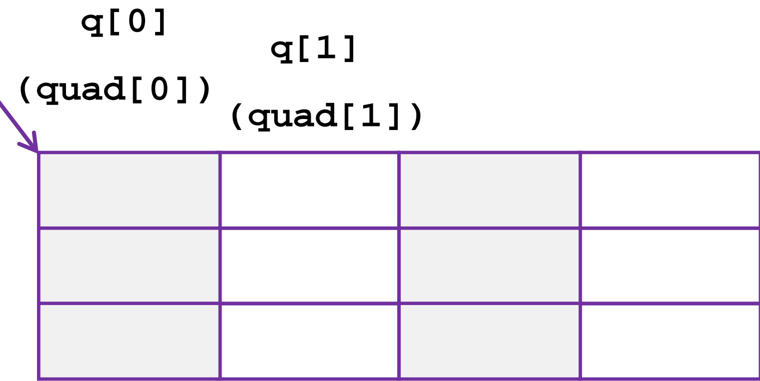


quad

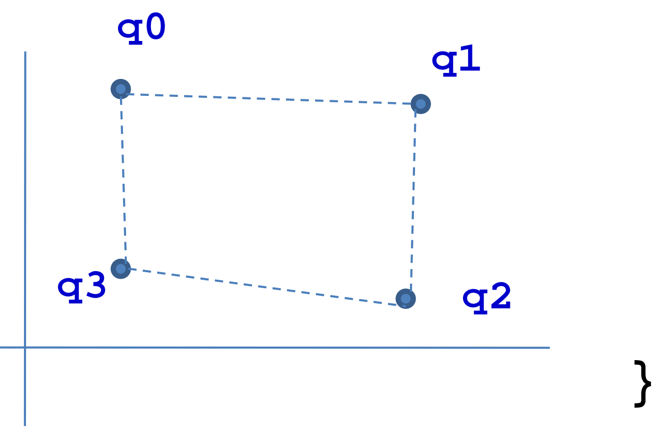

```
if (equi) printf ("...");
if (iso) printf ("...");
if (iso && equi)
      printf("...");

printf("\nFINE\n");
return 0;
}
```

Quadrilatero - funzioni - 1/5

```
void leggiQuadrilatero ( TipoPunto *q  ) {
int i;

 for (i=0; i<4; i++) {

                    ☺

 }
return;}
```

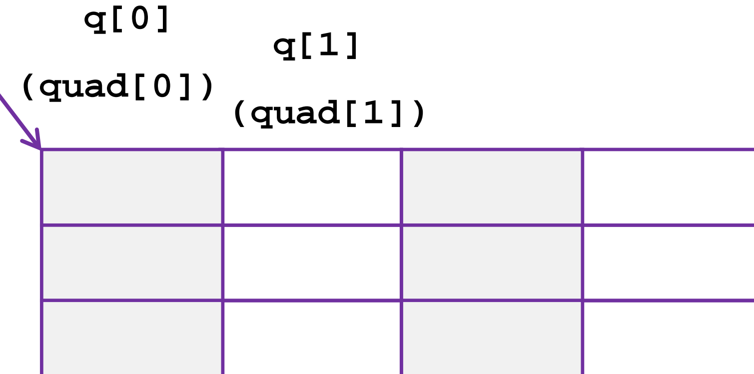



```
int equilatero (TipoPunto *q ) {   ☺

}
```



Quadrilatero - funzioni - 2/5

```
void leggiQuadrilatero ( TipoPunto *q  ) {
int i;

 for (i=0; i<4; i++) {
```

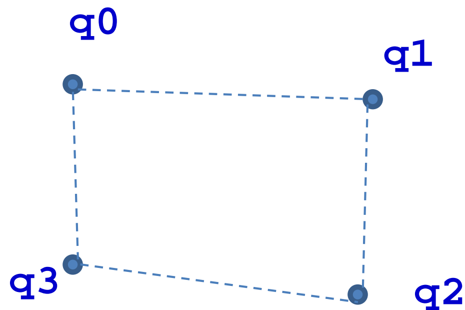



```
 }
return;}
```

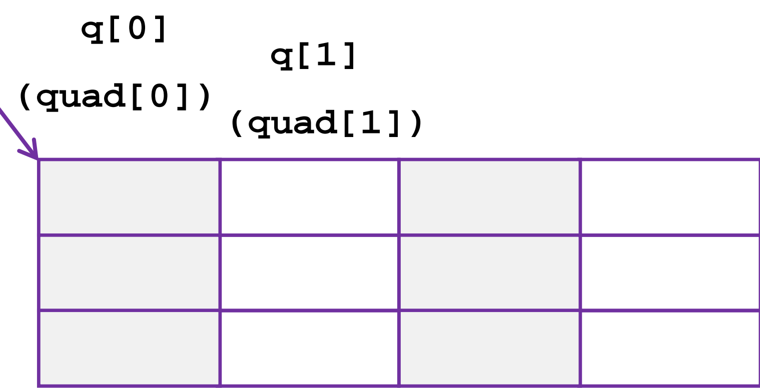
```
int equilatero (TipoPunto *q ) {
  double d = lunghezza(q[0], q[1]);
```

(lunghezza() e` una funzione che calcoli la distanza tra due punti passati come argomenti della chiamata)

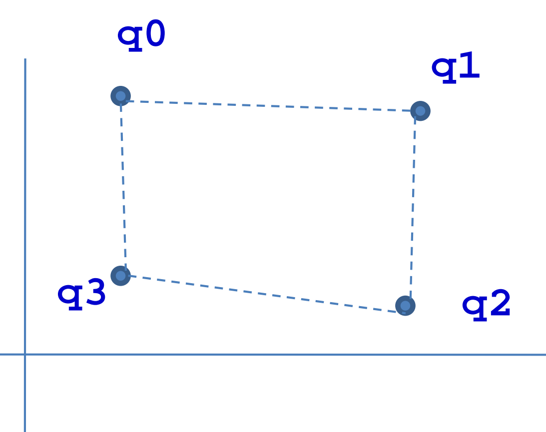
```
}
```



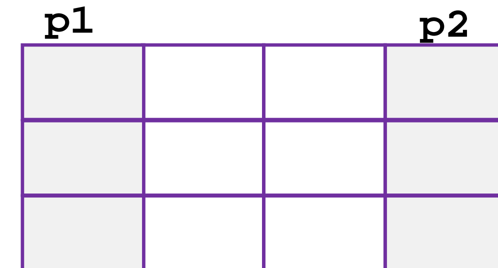
```
void leggiQuadrilatero ( TipoPunto *q  ) {
int i;

 for (i=0; i<4; i++) {
   printf("punto %d:\n ascissa ", i);
   scanf(("%lf", &q[i].ascissa);
   printf("punto %d:\n ordinata", i);
   scanf(("%lf", &q[i].ordinata);
   printf("punto %d:\n colore ", i);
   scanf(("%s", q[i].colore);
 }
return;}
```




```
int equilatero (TipoPunto *q ) {
  double d = lunghezza(q[0], q[1]);

 if (lunghezza(q[1], q[2]) != d)
   return 0;
 if (lunghezza(q[2], q[3]) != d)  return 0;
 if (lunghezza(q[3], q[0]) != d)  return 0;
return 1;
}
```

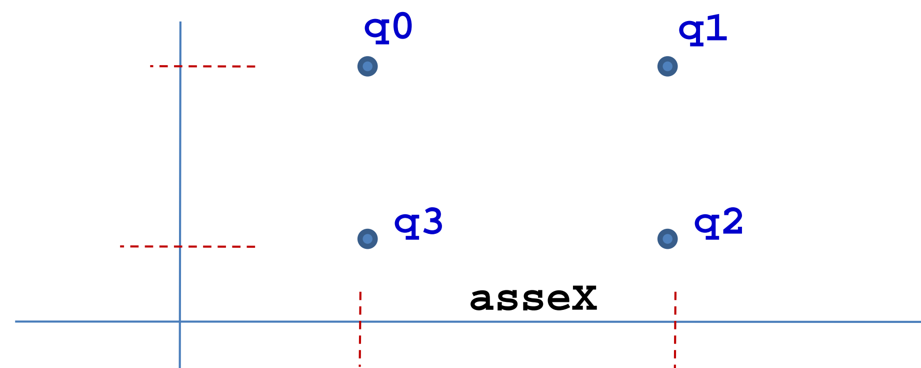


```
double lunghezza ( TipoPunto p1, TipoPunto p2 ) {
  /* ci vuole <math.h> */

 return (
   sqrt(
    pow(p1.ascissa - p2.ascissa, 2) +
    pow(p1.ordinata - p2.ordinata, 2)));
}
```

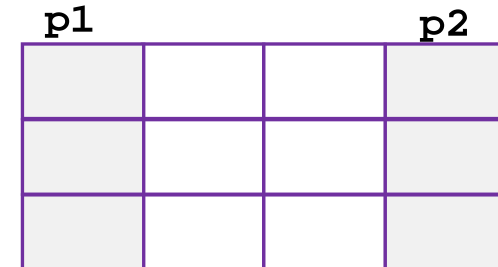



```
int isotetico(TipoPunto *q ) {

return (
              ☺
                      );
}
```

deve essere
q[0]q[1] // asseX
q[1]q[2] // asseY
q[2]q[3] // asseX
q[3]q[0] // asseY



```
double lunghezza ( TipoPunto p1, TipoPunto p2 ) {
  /* ci vuole <math.h> */

 return (
   sqrt(
    pow(p1.ascissa - p2.ascissa, 2) +
    pow(p1.ordinata - p2.ordinata, 2)));
}
```




```
int isotetico(TipoPunto *q ) {

return (
 (q[0].ordinata == q[1].ordinata) &&
 (q[1].ascissa == q[2].ascissa) &&
 (q[2].ordinata == q[3].ordinata) &&
 (q[3].ascissa == q[0].ascissa) );
}
```

```
deve essere
q[0]q[1] // asseX
q[1]q[2] // asseY
q[2]q[3] // asseX
q[3]q[0] // asseY
```