TIPO (Richiamo)

Un TIPO denota

(denota entrambe le seguenti cose ...)

COSTRUTTORI DI TIPO

NOMI DI TIPO

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

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:

Analogamente `typedef char Stringa[20]`

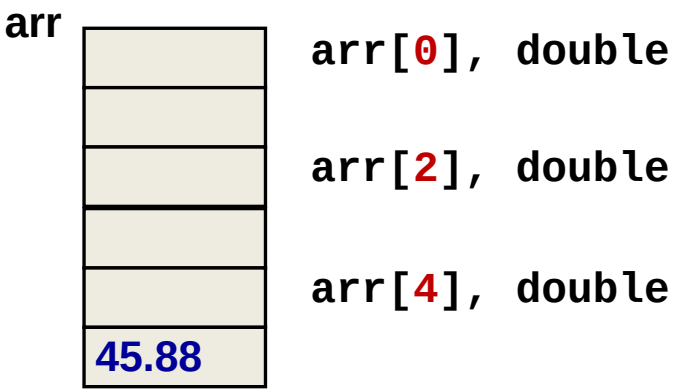
struct

`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**



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)

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

(2)

```
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**

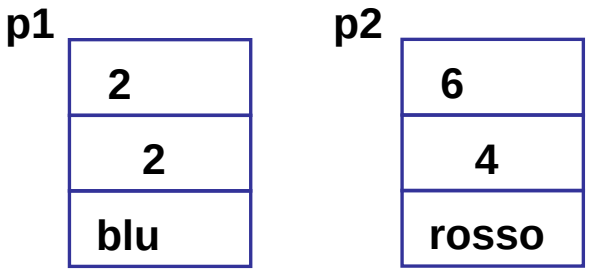
```
scanf("%s", p1.colore);
```

- **stampa del punto p1**

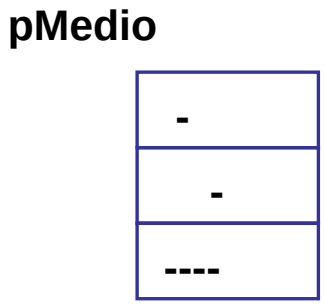
```
printf("(%g, %g):%s", p1.ascissa, p1.ordinata, p1.colore);
```

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)




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

/* successivamente riempiti */


/* il punto medio */
```



Algoritmo? ☺

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

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

```
Algoritmo …

0) ci servono




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

2)  stampa
```

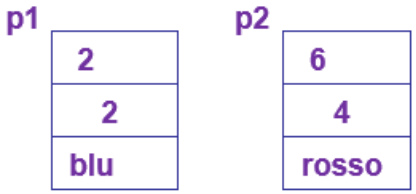
primo uso di struct, punto colorato medio - 3/3 -

```
#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);
...
```

dichiarazione del tipo effettuata a TOP LEVEL

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

```
...
  pMedio.ascissa = (p1.ascissa + p2.ascissa)/2;        /*      */
                                                       /*      */
  if (strcmp(p1.colore, p2.colore)!=0)                 /*      */

  else
```




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

```
#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()???

varie sulle struct ...

inizializzazione di una struct in definizione

```
struct punto {  ...
};
```

definizione alternativa di 2 variabili struct

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

definizione alternativa di 2 var struct, con TIPO ANONIMO

```
struct



} p1, p2;
/* dopo, non possiamo definirne altre */
```

```
typedef




/* 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]
};
```

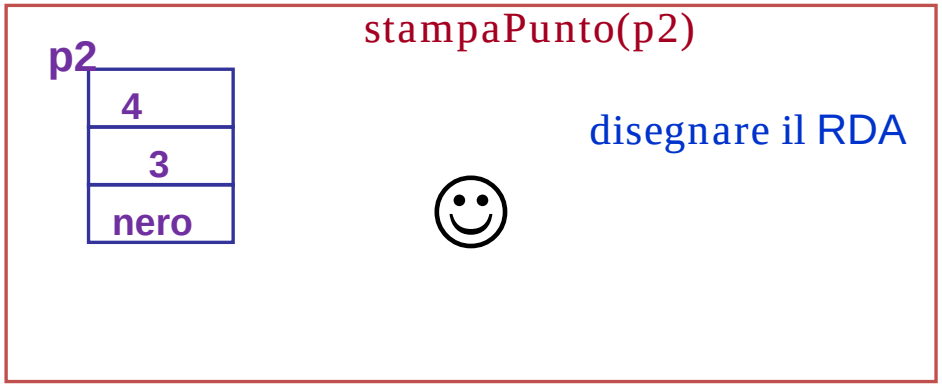
```
int main() {
    TipoPunto p1, p2, pMedio;

  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;
}
```

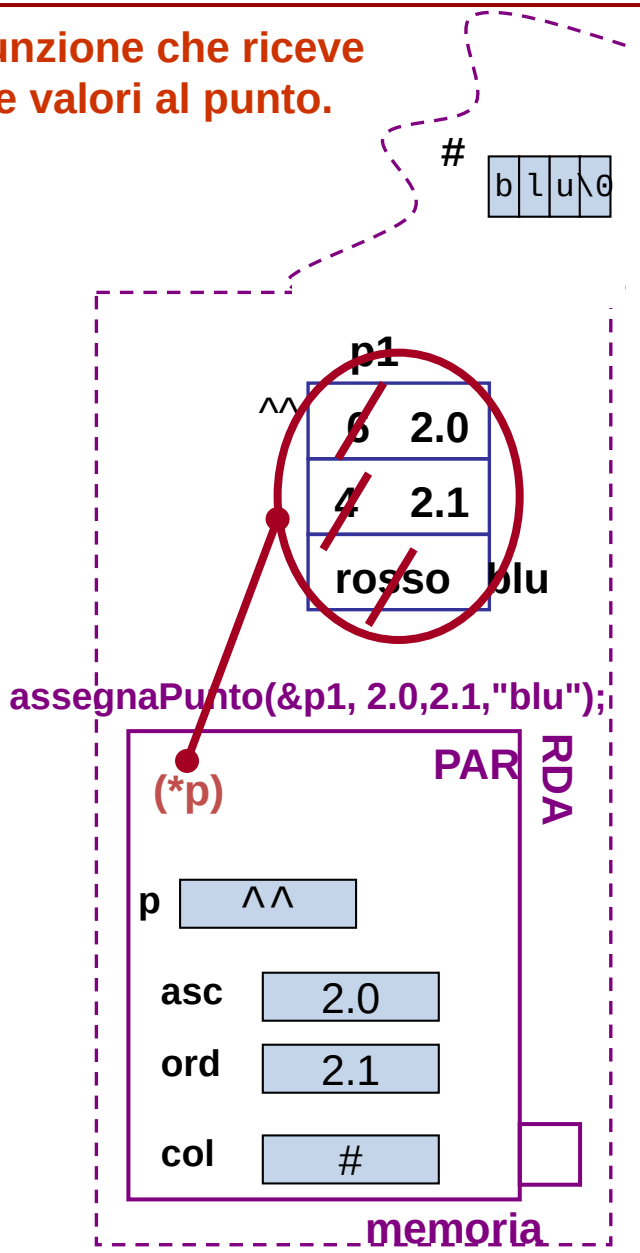


Funzioni con side effect su struct - 1/3 -

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={...};

  ...

  ...
return 0;
}
```

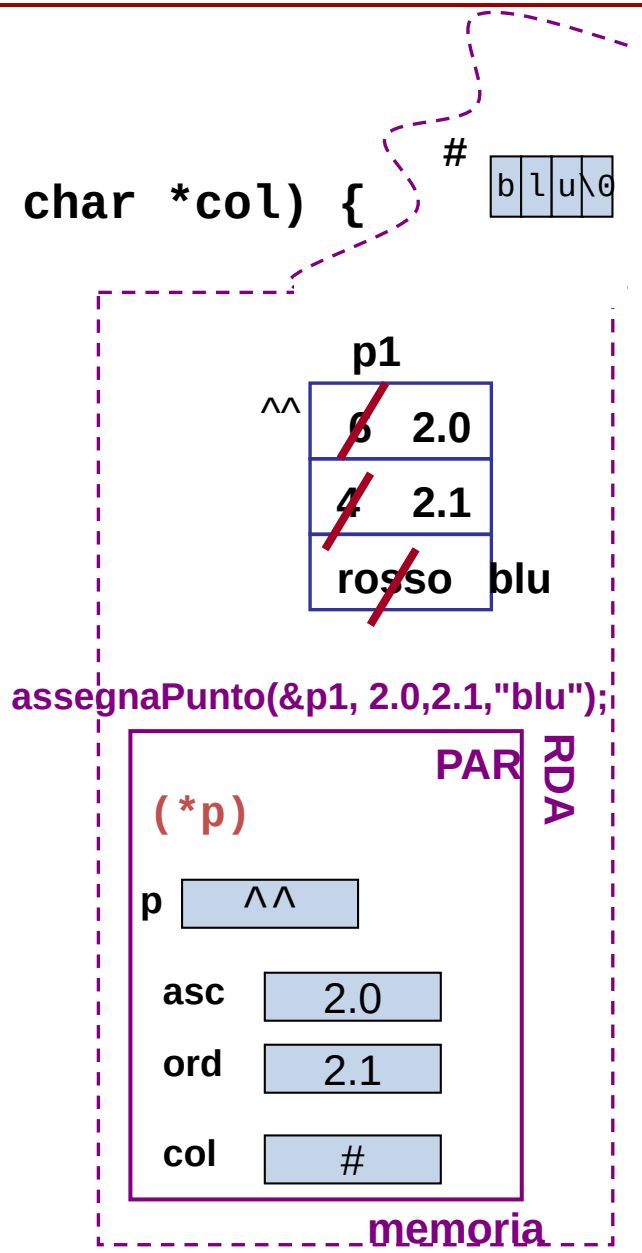


come e` definite assegnaPunto? ☺

Funzioni con side effect su struct - 2/3 -

Funzioni che provocano side effect su struct

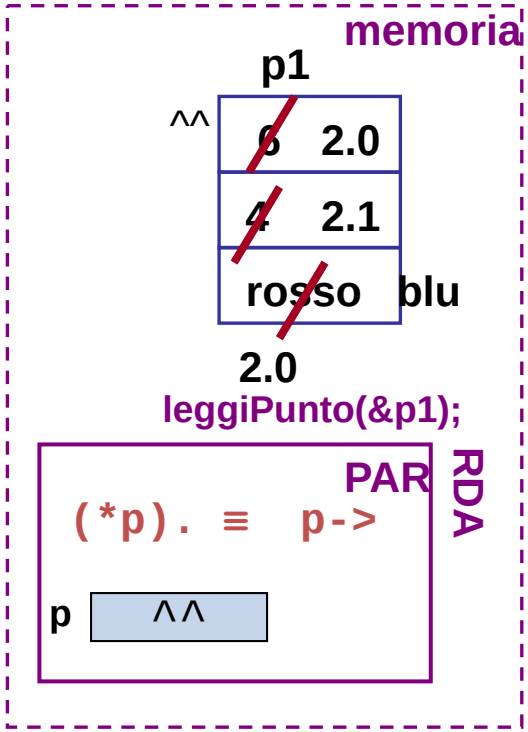
```
void assegnaPunto ( struct punto *p,
                    double asc, double ord, char *col) {
   (*p).ascissa = asc;

   strcpy((*p).colore, col);

return;
}
```



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) {




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;
}
```

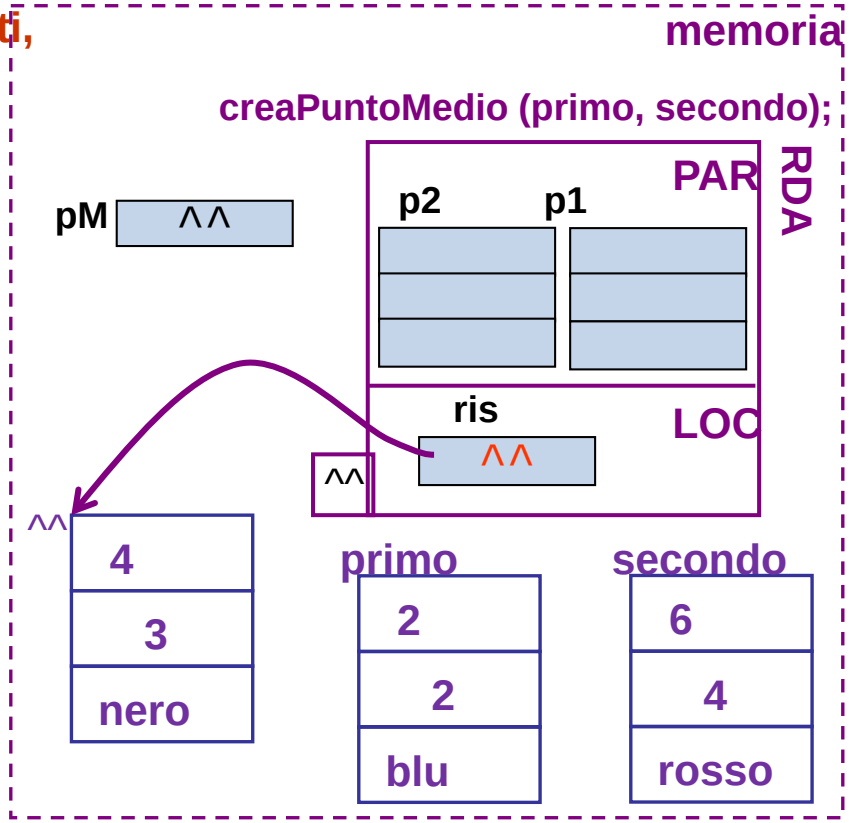
Funzioni che restituiscono puntatori a struct - 1/5 -

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;
}
```

☺ Parametro attuale di stampaPunto ?

Intestazione della definizione di creaPuntoMedio ?

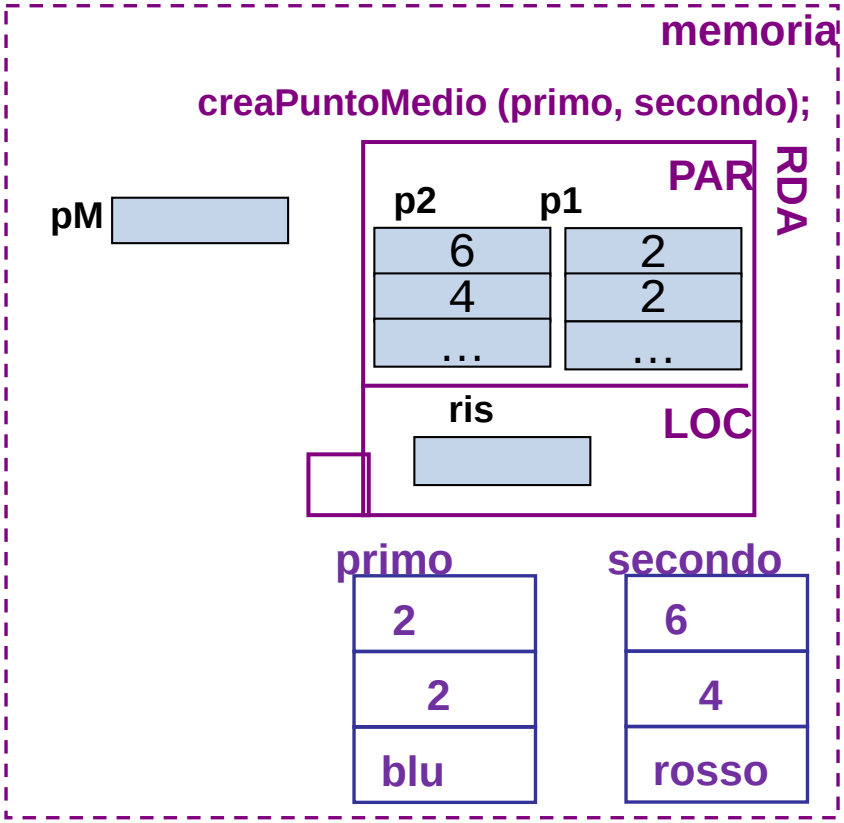


☺

Funzioni che restituiscono puntatori a struct – 5/5 –

Funzioni che restituiscono un puntatore a struct

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

  TipoPunto *ris;



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



   if (strcmp(p1.colore, p2.colore))
     strcpy(ris->colore, "nero");
   else strcpy(ris->colore, p2.colore);
 }
return ris;
}
```

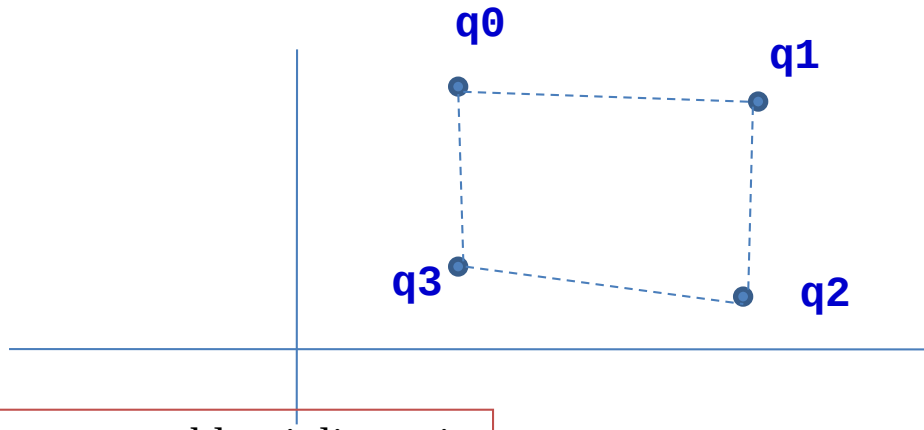


Quadrilatero

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

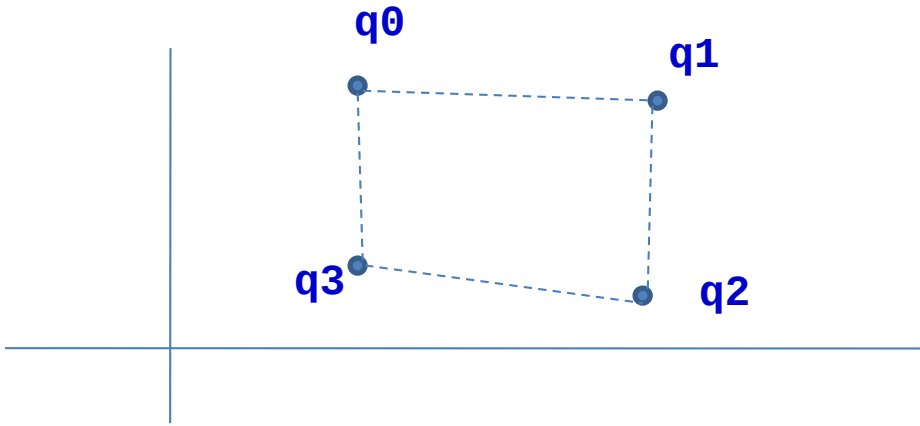
- algoritmo,
- strutture dati coinvolte,
- main(), con chiamate di funzioni di servizio (es. sottoprogrammi per sottoproblemi)
- definizioni delle funzioni



NB usiamo TipoPunto invece che TipoPuntoColorato, per problemi di spazio

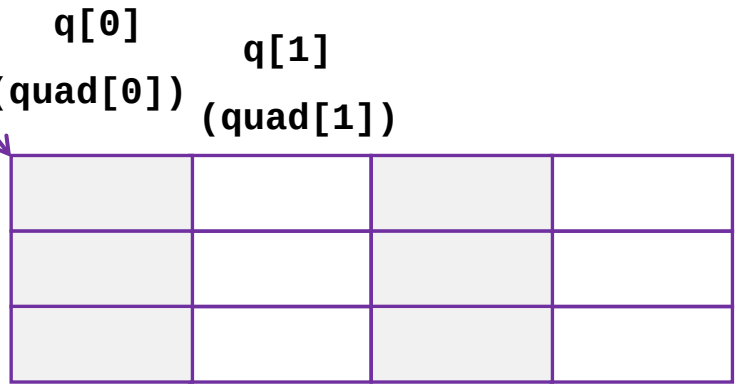
Quadrilatero - la struttura dati

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

int main() {  TipoPunto quad  ?

   int iso, equi, i;
```



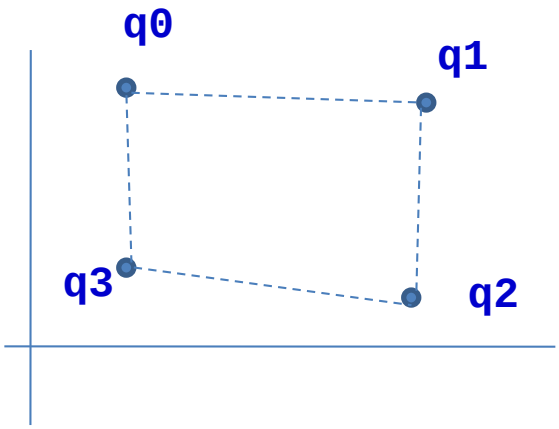
Quadrilatero - funzioni - 1/5

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

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

 }
return;}
```



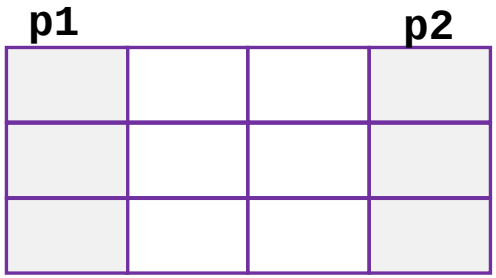
☺

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

}
```



Quadrilatero - funzioni - 4/5

```
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