

## Esercitazioni Guidate di Tecniche della Programmazione

Note introduttive: Seguire le raccomandazioni date nelle precedenti EG ...

### 6. Esercitazione GUIDATA 6

#### 6.1. Uso di *struct* (*punto.c*, *punto2.c*).

Scrivere un programma in cui

- sia definito il tipo `struct punto` adatto a rappresentare i punti colorati nel piano cartesiano (due dimensioni);
- vengano lette coordinate e colori di due punti e venga costruito il punto intermedio, stampandone i dati (il punto intermedio ha il medesimo colore dei due punti letti, se questi hanno il medesimo colore; altrimenti ha colore "NERO").

Scrivere poi il medesimo programma, in cui il tipo dei punti colorati sia `TipoPunto`, definito mediante `typedef` ed usato nelle dichiarazioni dei punti usati nel programma.

#### 6.2. Uso di *struct* (*distanze.c*).

Scrivere un programma in cui viene letta una sequenza di punti colorati (definiti come negli esercizi precedenti) e, per ogni punto letto, dal secondo in poi, viene stampata la *distanza colorata* tra esso e il precedente.

Ci sono dei colori prestabiliti come ammissibili: "bianco", "rosso", "giallo", "celeste", "blu", "NERO"

Quando viene letto il colore di un punto, se questo è al di fuori dei colori prestabiliti, sarà considerato "NERO".

La *distanza colorata* tra due punti è definita come la coppia

$\langle dl, dc \rangle$ , dove

- $dl$ =distanza lineare tra i due punti e
- $dc$ =distanza tra i colori dei due punti (stabilita in base alla definizione dei colori prestabiliti data in precedenza: ad es. la distanza tra bianco e rosso è 1, quella tra bianco e giallo è 2, quella tra giallo e nero è 3).

## Qualche suggerimento per la soluzione segue: non li leggete tutti d'un fiato ...

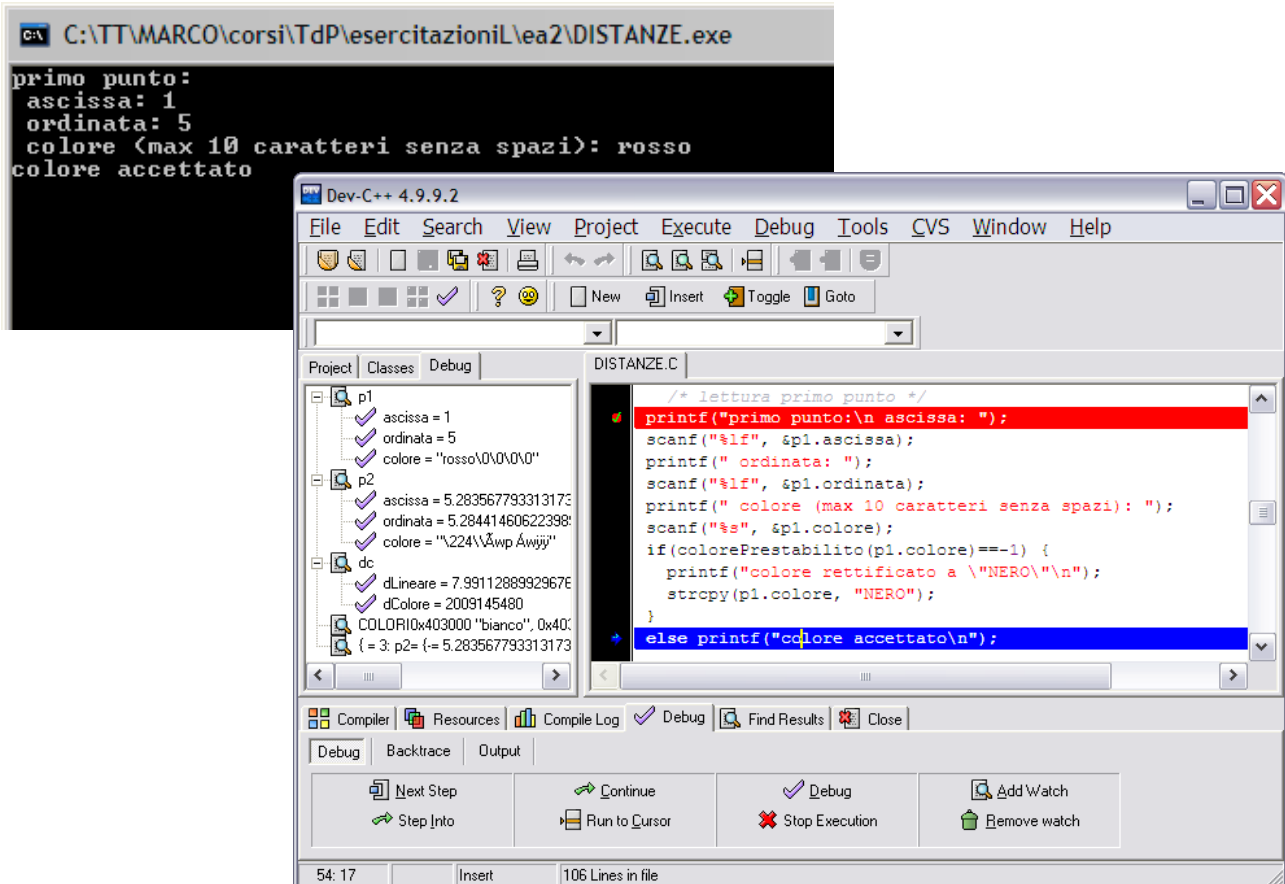
### Suggerimento 0

Quando il programma compila, probabilmente siamo solo a meta' dell'opera. Provare il programma anche usando le funzionalità di debugging:

#### debugging?            What's *debugging*??

- la linea rossa che si vede in figura è una linea su cui è stato impostato un breakpoint (basta cliccare accanto alla linea per impostare il breakpoint o per rimuoverlo).
- Un breakpoint nel punto indicato è utile: così' possiamo scegliere l'opzione di compilazione con *debug* (F8) e usare F7 (*Next Step*) per far avanzare l'esecuzione del programma passo passo (un'istruzione alla volta).
- Durante l'esecuzione passo passo è utile tener d'occhio il contenuto di alcune variabili significative: in figura si vede che è stata impostata la visualizzazione del contenuto delle variabili p1, p2, dc: è stato usato il comando *Add Watch*. (Nell'aggiungere una nuova watch, può essere necessario eseguire il passo successivo – F7– prima di vederla effettivamente.).
- La figura mostra un momento dell'esecuzione in cui tutte le istruzioni di lettura del punto p1 sono state eseguite (la watch su p1 mostra quei valori: 1, 5, rosso).

- Se una watch corrisponde a variabili non nello scope dell'istruzione in esecuzione, ciò viene indicato (ad esempio nel punto del programma in esecuzione non potremmo vedere il contenuto della variabile `i` definita nella funzione `colorePrestabilito()`).
- Quando è in esecuzione un'istruzione di lettura, bisogna andare nella finestra di esecuzione e inserire il dato da leggere ...
- Se non si vuole più procedere passo passo, `Shift F7 (Continue)` procura l'avanzamento dell'esecuzione fino alla prossima occorrenza di un breakpoint.



## Suggerimento 1di5

la struttura del TipoPunto è quella già vista; inoltre potrebbe convenire dare una definizione di un tipo apposito per la DistanzaColorata. (una struct con un campo double per la distanza lineare e uno int per la distanza tra i colori).

Inoltre è bene che sia data una chiara definizione dei colori prestabiliti. Questa potrebbe anche non essere la definizione di un tipo vero e proprio. Una possibilità è quella di usare un array top level di stringhe inizializzato opportunamente `COLORI[] = {bianco, rosso, giallo, celeste, blu, nero}`.

### **Suggerimento 2di5**

All'array `COLORI` potrebbe essere affiancata una funzione `colorePrestabilito` che, ricevendo un colore, `col`, dica se `col` è o no uno di colori ammessi. Nella soluzione proposta questa è una funzione intera che restituisce `-1` se `col` non è ammissibile, oppure l'indice di `col` nell'array `COLORI`.

### **Suggerimento 3di5**

La funzione `colorePrestabilito` (sempre nella soluzione proposta) viene usata anche per calcolare la seconda parte della distanza colorata (infatti se il colore di  $p_1$  è  $k_1$  e il colore di  $p_2$  è  $k_2$ , la distanza tra i colori sarà  $k_1 - k_2$  o  $k_2 - k_1$  ...)

### Suggerimento 4di5

Schema del programma

- lettura primo punto ( $p_1$ )
- ciclo del tipo 

```
while (continua!=0) {  
    ...  
}
```

con richiesta finale all'utente di inserire 1 se vuole continuare o 0 se vuole terminare (scelta che viene letta in `continua`).

### Suggerimento 5di5

- durante ogni iterazione del ciclo
  - o si legge il punto  $p_2$  (si leggono le sue coordinate e colore)
  - o si calcola la distanza colorata tra  $p_1$  e  $p_2$  (e la si stampa)

- l'utente indica la sua scelta (se continuare o meno) e di conseguenza, in accordo con questa scelta, viene assegnata la variabile `continua`.
- Infine viene copiato `p2` in `p1`, così in `p1` ci sarà l'ultimo punto letto da input, e se ne potrà calcolare la distanza da quello successivamente inserito

### **6.3. Funzioni che ricevono e/o restituiscono `struct` (punto3.c).**

Riscrivere il programma del punto 1.1, definendo ed usando le seguenti funzioni:

```
/*funzione che riceve un punto e lo stampa */
void stampaPunto(struct Punto p)

/* funzione che riceve due punti e ne restituisce
   il punto medio */
struct Punto puntoMedio(struct Punto pr, struct Punto sec)
```

### **6.4. Funzioni che ricevono puntatori a `struct` (punto4.c).**

Scrivere un programma che soddisfi i seguenti requisiti:

- il programma legge i dati relativi a due punti colorati (definiti come in precedenza)
- il programma calcola e stampa i dati relativi al punto medio tra i due letti da input;
- poi il programma chiede in input un colore e assegna tale colore a tutti e tre i punti, stampandone successivamente i dati in output.

Il programma deve far uso delle funzioni definite nell'esercizio precedente e della funzione

```
void cambiaColoreA(TipoPunto *p, char col[])
```

che, ricevendo un punto (o meglio, il suo indirizzo), e un colore (`col`), modifica il punto assegnandogli `col` come colore. Suggestioni seguono.

### **Suggerimento:**

conviene prendere come punto di partenza il programma fatto per l'esercizio precedente, aggiungendo la nuova funzione e modificando opportunamente la `main()`.

### **6.5. Funzioni che ricevono puntatori a struct (punto5.c).**

Ripetere l'esercizio del 2.3, definendo e utilizzando, per la lettura di ciascun punto, una funzione `leggiPunto(...)`

**Suggerimento 1di2:** la funzione deve ricevere un punto e riempirlo con dati letti da input.

**Suggerimento 2di2:** quindi la funzione deve ricevere l'indirizzo del punto da riempire.

### **6.6. Funzioni che restituiscono puntatori a struct (punto6.c).**

Rifacendosi a quanto fatto in precedenza, scrivere una funzione `creaPuntoMedio(...)` che, ricevendo due punti, restituisce una struct appositamente allocata dinamicamente, contenente la rappresentazione del punto medio tra i due parametri.

**Suggerimento 1di2:**

Si tratta di rifare l'esercizio precedente, ma sostituendo la funzione puntoMedio con quella qui richiesta (che restituisce un puntatore a struct Punto e non una struct Punto).

### Suggerimento 2di2:

ecco uno stralcio del programma PUNTO6.C in cui si vede come viene usata la funzione creaPuntoMedio:

```
int main(){
    TipoPunto p1, p2,
        *pM;    /* puntatore per il punto medio */

    printf("primo punto:");
    leggiPunto(&p1);
    printf("secondo punto:");
    leggiPunto(&p2);

    pM = creaPMedio(p1, p2);
    /* la chiamata a pMedio restituisce il puntatore ad una struct che e'
    stata allocata appositamente, e che rappresenta il punto medio tra p1 e
    p2 */

    printf(" - adesso i tre punti sono:\n - p1 = ");
```



### 6.7. Quadrilateri (*quadri.c*).

L'oggetto di questo esercizio è un programma capace di gestire quadrilateri dati in input.

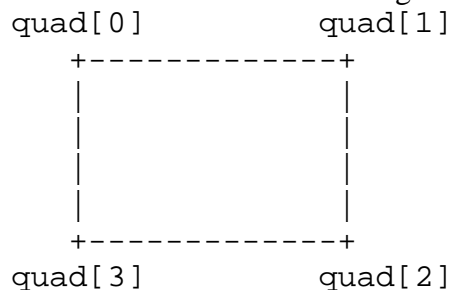
Un quadrilatero viene specificato come l'insieme dei suoi quattro vertici, che sono punti colorati sul piano (cioè variabili del tipo `TipoPunto` definito precedentemente).

Supponiamo che i punti dati in input siano tutti distinti e diano luogo ad una figura in cui i lati sono "a 90 gradi".

Cio' assunto, il programma

- costruisce un quadrilatero allocando dinamicamente un array di quattro punti e leggendo i dati relativi ai quattro vertici;
- verifica se il quadrilatero è un quadrato
- verifica se il quadrilatero è isotetico, cioè i suoi lati sono paralleli agli assi cartesiani;
- stampa i dati dei quattro vertici e stampa un messaggio in cui sia specificato se il quadrilatero è o no un quadrato ed è o no isotetico.

Se `quad` è l'identificatore usato per il quadrilatero nel programma, si assume anche che i punti siano distribuiti come nel disegno qui sotto:



Usare nel programma le seguenti funzioni:

- `void stampaPunto(TipoPunto )` (per stampare i dati di un punto passato per parametro)
- `void leggiPunto( TipoPunto *)` (per leggere un punto)
- `void leggiQuadrilatero(TipoPunto q[4])` (per leggere l'intero quadrilatero, usando `leggiPunto`);
- `void stampaQuadrilatero(TipoPunto q[4])` (per stampare i dati del quadrilatero, ad esempio come sequenza dei punti che ne sono vertici);
- `double lunghezza(TipoPunto primo, TipoPunto secondo)` (per calcolare la distanza tra due punti)

### **Suggerimento 1di2:**

per verificare che il quadrilatero `quad` sia un quadrato basta verificare che siano uguali le misure dei suoi lati (cioè delle distanze tra vertici consecutivi: `quad[0]--quad[1]`, `quad[1]--quad[2]`, `quad[2]--quad[3]`, `quad[3]--quad[0]`),).

Per calcolare la lunghezza del lato `quad[k]quad[h]` si usa la funzione `lunghezza(...)`.

### **Suggerimento 2di2:**

per verificare che i lati sono paralleli agli assi, bisogna verificare che i lati

- `quad[0]--quad[1]` e `quad[2]--quad[3]` sian paralleli all'asse delle ascisse
- `quad[1]--quad[2]` e `quad[3]--quad[0]` sian paralleli all'asse delle ordinate.

### **6.8. Poligoni**

Scrivere un programma in cui viene ottenuto da input un poligono regolare, e viene prodotto un nuovo poligono, i cui vertici sono i punti medi dei lati del poligono di partenza.

(Come rappresentare un poligono? Come rappresentare il poligono risultante? Ci aiuta l'esercizio precedente ...)