Programmazione 1

Lezione 9

Vincenzo Marra

vincenzo.marra@unimi.it

Dipartimento di Matematica *Federigo Enriques*
Università degli Studi di Milano

10 maggio 2017

Le strutture

Le strutture permettono di accorpare in una singola unità sintattica tipi di dati compositi. Per esempio, possiamo denotare un punto del piano reale con una coppia di `double`, le sue coordinate. Per accorpare le due coordinate in una singola unità, dichiariamo:

```
1  struct punto //Etichetta (opzionale) della struttura
2  {
3          double x; //Primo membro
4          double y; //Secondo membro
5  };
```

Le strutture

Le strutture permettono di accorpare in una singola unità sintattica tipi di dati compositi. Per esempio, possiamo denotare un punto del piano reale con una coppia di `double`, le sue coordinate. Per accorpare le due coordinate in una singola unità, dichiariamo:

```
1 struct punto //Etichetta (opzionale) della struttura
2 {
3         double x; //Primo membro
4         double y; //Secondo membro
5 };
```

Fatto centrale. Queste righe di codice definiscono un nuovo tipo (senza allocare memoria).

Le strutture

Le strutture permettono di accorpare in una singola unità sintattica tipi di dati compositi. Per esempio, possiamo denotare un punto del piano reale con una coppia di `double`, le sue coordinate. Per accorpare le due coordinate in una singola unità, dichiariamo:

```
1 struct punto //Etichetta (opzionale) della struttura
2 {
3         double x; //Primo membro
4         double y; //Secondo membro
5 };
```

Fatto centrale. Queste righe di codice definiscono un nuovo tipo (senza allocare memoria). Esso è il tipo `struct punto`.

Le strutture

Le strutture permettono di accorpare in una singola unità sintattica tipi di dati compositi. Per esempio, possiamo denotare un punto del piano reale con una coppia di `double`, le sue coordinate. Per accorpare le due coordinate in una singola unità, dichiariamo:

```
1 struct punto //Etichetta (opzionale) della struttura
2 {
3         double x; //Primo membro
4         double y; //Secondo membro
5 };
```

Fatto centrale. Queste righe di codice definiscono un nuovo tipo (senza allocare memoria). Esso è il tipo `struct punto`. Se l'etichetta `punto` — che è un'abbreviazione di ciò che la segue, fino alla graffa di chiusura — è assente, il nome del nuovo tipo va da `struct` (incluso) alla graffa di chiusura.

Dichiarazioni di variabili

Segue che è legittimo e sensato dichiarare:

```
struct punto p; //Un punto (non inizializzato)
```

Dichiarazioni di variabili

Segue che è legittimo e sensato dichiarare:

```
struct punto p; //Un punto (non inizializzato)
```

Fatto centrale. Questa dichiarazione alloca memoria sufficiente a memorizzare tutti i membri della struttura (in questo esempi, i membri sono due `double`).

Dichiarazioni di variabili

Segue che è legittimo e sensato dichiarare:

```
struct punto p; //Un punto (non inizializzato)
```

Fatto centrale. Questa dichiarazione alloca memoria sufficiente a memorizzare tutti i membri della struttura (in questo esempi, i membri sono due `double`).

Nota che è anche legittimo e sensato il brano seguente, che codifica dichiarazione e definizione assieme, senza etichetta:

```
1 struct
2 {
3         double x; //Primo membro
4         double y; //Secondo membro
5 } p,q; //Due punti (non inizializzati)
```

Inizializzazione, assegnazioni, parametri

Le variabili di tipo `struct ...` si possono inizializzare, al momento della dichiarazione, con liste di valori costanti, in modo simile a quanto abbiamo visto per gli array. Per esempio:

```
struct punto q = {-0.1, 2}; //Inizializzazione con lista
```

Inizializzazione, assegnazioni, parametri

Le variabili di tipo `struct ...` si possono inizializzare, al momento della dichiarazione, con liste di valori costanti, in modo simile a quanto abbiamo visto per gli array. Per esempio:

```
struct punto q = {-0.1, 2}; //Inizializzazione con lista
```

Tali variabili possono anche essere assegnate. Così, è legittimo:

```
1  struct punto //Etichetta (opzionale) della struttura
2  {
3          double x; //Primo membro
4          double y; //Secondo membro
5  };
6  struct punto p; //Un punto (non inizializzato)
7  struct punto q = {-0.1, 2}; //Inizializzazione con lista
8  p=q; //Assegna q a p
```

Inizializzazione, assegnazioni, parametri

Si possono poi passare le variabili automatiche di tipo `struct ...` come parametri alle funzioni. E le funzioni possono restituire valori di un tale tipo strutturato.
Per esempio:

funzstrutt.c

```
1  /* Definizione globale */
2  struct punto //Etichetta (opzionale) della struttura
3  {
4          double x; //Primo membro
5          double y; //Secondo membro
6  };
7
8  /* Nel main */
9  struct punto O = {0.0, 0.0}; //Un punto
10 struct punto p = {1.0, 1.0}; //Un altro punto
11 struct punto m;//Un terzo punto (non inizializzato)
12 m=medio(O,p); //restitusice il punto medio degli args
```

Inizializzazione, assegnazioni, parametri

Si possono poi passare le variabili automatiche di tipo `struct {...}` come parametri alle funzioni. E le funzioni possono restituire valori di un tale tipo strutturato.

Il prototipo della funzione `medio` deve allora essere:

```
struct punto medio(struct punto, struct punto);
```

Nota. Anche qui, *come sempre in C*, il passaggio dei parametri avviene per copia. Naturalmente è possibile dichiarare e passare come argomenti puntatori a strutture. Per esempio, la dichiarazione

```
struct punto *pnt;
```

dichiara (ma non inizializza) un puntatore a `struct punto`.

Accesso ai campi

Per accedere a un membro di una struttura si usa la sintassi

```
nome-strutt.membro-strutt
```

Così:

Accesso ai campi

Per accedere a un membro di una struttura si usa la sintassi

`nome-strutt.membro-strutt`

Così:

```
1  struct
2  {
3          double x; //Primo membro
4          double y; //Secondo membro
5  } p; //Un punto (non inizializzato)
6
7  p.x=0.0; //Assegna ascissa
8  p.y=1.1; //Assegna ordinata
```

Campo di visibilità dei membri e *shadowing*

Il nome completo del membro ascissa è dunque `p.x`, e non solo `x`. La coppia di dichiarazioni:

Campo di visibilità dei membri e *shadowing*

Il nome completo del membro ascissa è dunque `p.x`, e non solo `x`. La coppia di dichiarazioni:

```
10 struct punto p;
11 int x;
```

non dà quindi luogo ad alcun effetto di *shadowing*: non vi sono ambiguità possibili fra

`p.x`

che qui è la variabile di tipo `double`, e

`x`

che qui invece è la variabile di tipo `int`.

Strutture innestate

Un membro di una struttura può essere esso stesso una struttura. Supponiamo di codificare un triangolo nel piano tramite i suoi tre vertici. Così:

Strutture innestate

Un membro di una struttura può essere esso stesso una struttura. Supponiamo di codificare un triangolo nel piano tramite i suoi tre vertici. Così:

```
1 struct tri
2 {
3         struct punto a; //Primo membro
4         struct punto b; //Secondo membro
5         struct punto c; //Terzo membro
6 } t; //Un triangolo (non inizializzato)
```

Strutture innestate

Un membro di una struttura può essere esso stesso una struttura. Supponiamo di codificare un triangolo nel piano tramite i suoi tre vertici. Così:

```
1  struct tri
2  {
3          struct punto a; //Primo membro
4          struct punto b; //Secondo membro
5          struct punto c; //Terzo membro
6  } t; //Un triangolo (non inizializzato)
```

L'accesso ai singoli vertici si codifica così:

```
8  t.a={0.0,1.0}; //Inizializzazione del vert a di t
```

Strutture innestate

Un membro di una struttura può essere esso stesso una struttura. Supponiamo di codificare un triangolo nel piano tramite i suoi tre vertici. Così:

```
1  struct tri
2  {
3          struct punto a; //Primo membro
4          struct punto b; //Secondo membro
5          struct punto c; //Terzo membro
6  } t; //Un triangolo (non inizializzato)
```

L'accesso alle coordinate dei vertici si codifica così:

```
9  t.b.x=-1.0; //Inizializzazione dell'ascissa del vert b di t
```

Esercizio

Si scriva un programma che legga dalla riga di comando le coordinate di due punti nel piano, e restituisca le coordinate del loro punto medio. Si usino le strutture per rappresentare i punti del piano.

ptmedio.c

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
        if (argc < 5)
        {
                printf("Errore: mancano argomenti.\n");
                return -1;
        }
        struct pt
        {
                double x;
                double y;
        };
        struct pt p,q;
        p.x=atof(argv[1]); p.y=atof(argv[2]);
        q.x=atof(argv[3]); q.y=atof(argv[4]);

        struct pt m = { (p.x+q.x)/2, (p.y+q.y)/2 };

        printf("Il punto medio di (%g,%g) e (%g,%g) e' (%g,%g).\n", p.x,p.y,
                q.x,q.y,m.x,m.y);
        return 0;
}
```

Puntatori alle strutture e operatore ->

I puntatori alle strutture sono del tutto analoghi ai puntatori alle variabili ordinarie. Così:

```
1  struct tri
2  {
3          struct punto a;
4          struct punto b;
5          struct punto c;
6  };
7
8  struct tri s, t = {{0,0}, {0,1}, {1,0}}; //Due triangoli
9  struct tri *pt; //Puntatore a struct tri, non inizializzato
10 pt=&t; //pt punta a t
```

Puntatori alle strutture e operatore ->

I puntatori alle strutture sono del tutto analoghi ai puntatori alle variabili ordinarie. Così:

```
1  struct tri
2  {
3          struct punto a;
4          struct punto b;
5          struct punto c;
6  };
7
8  struct tri s, t = {{0,0}, {0,1}, {1,0}}; //Due triangoli
9  struct tri *pt; //Puntatore a struct tri, non inizializzato
10 pt=&t; //pt punta a t
```

Accesso al vertice a di t:

```
11 s.a=(*pt).a; //accede al vert a di t. Parentesi necessarie.
```

Puntatori alle strutture e operatore ->

I puntatori alle strutture sono del tutto analoghi ai puntatori alle variabili ordinarie. Così:

```
1  struct tri
2  {
3          struct punto a;
4          struct punto b;
5          struct punto c;
6  };
7
8  struct tri s, t = {{0,0}, {0,1}, {1,0}}; //Due triangoli
9  struct tri *pt; //Puntatore a struct tri, non inizializzato
10 pt=&t; //pt punta a t
```

Accesso alla coordinata x del vertice b di t:

```
12 s.b.x=(*pt).b.x; //accede all'ascissa del vert b di t.
```

Puntatori alle strutture e operatore ->

I puntatori alle strutture sono del tutto analoghi ai puntatori alle variabili ordinarie. Così:

```
1  struct tri
2  {
3          struct punto a;
4          struct punto b;
5          struct punto c;
6  };
7
8  struct tri s, t = {{0,0}, {0,1}, {1,0}}; //Due triangoli
9  struct tri *pt; //Puntatore a struct tri, non inizializzato
10 pt=&t; //pt punta a t
```

Abbreviazione ->. Accesso al vertice c di t:

```
13 s.c=pt->c; //accede al vert c di t.
```

Puntatori alle strutture e operatore ->

I puntatori alle strutture sono del tutto analoghi ai puntatori alle variabili ordinarie. Così:

```
1  struct tri
2  {
3          struct punto a;
4          struct punto b;
5          struct punto c;
6  };
7
8  struct tri s, t = {{0,0}, {0,1}, {1,0}}; //Due triangoli
9  struct tri *pt; //Puntatore a struct tri, non inizializzato
10 pt=&t; //pt punta a t
```

Abbreviazione ->. Accesso alla coordinata y del vertice c di t:

```
14 pt->c.y=2.5; //accede all'ordinata del vert c di t.
```

Esercizi

1. Si scriva una funzione che accetti in ingresso due puntatori a strutture che rappresentano punti nel piano, e restituisca un puntatore al punto di norma euclidea minima.
2. Si definisca una struttura atta a rappresentare libri, nella forma *Cognome Autore, Titolo, Data Pubblicazione*. Si scriva poi una funzione che accetti in ingresso un (puntatore a un) array di tali strutture, e restituisca l'indice del libro di pubblicazione più recente.

Segue una soluzione del primo esercizio. Il secondo sarà svolto in classe tempo permettendo, oppure potrete risolverlo in laboratorio.

norm.c

```c
#include <stdio.h>
#include <math.h>

struct punto
{
        double x;
        double y;
};

double norma(struct punto *);
struct punto *minnorm(struct punto *, struct punto *);

int main(void)
{
        struct punto p = {0,0}, q = {1,1};
        printf("Dei due punti (%g,%g) e (%g,%g),", p.x,p.y,q.x,q.y);
        struct punto *pt = minnorm(&p,&q);
        printf("il punto di norma minima e' (%g,%g).\n", pt->x,pt->y);

        return 0;
}

double norma(struct punto *a)
{
        return sqrt( (a->x)*(a->x)+(a->y)*(a->y) );
}

struct punto *minnorm(struct punto *a, struct punto *b)
{
        if (norma(a) < norma (b))
                return a;
        return b;
}
```