# Aggregati di dati eterogenei: il tipo struct

## Esercizi risolti

## 1 Esercizio

Si vuole realizzare un tipo struct utilizzato per informazioni su operazioni di vendita, avente i seguenti campi:

- codice: numero intero indicante il codice di riferimento dell'articolo venduto
- nome: stringa di lunghezza inferiore a 20 caratteri
- prezzo: numero reale (float) corrispondente ap prezzo unitario dell'articolo
- npezzi: numero (intero) di pezzi venduti

Di tale tipo si vogliono successivamente dichiarare due variabili scalari (x, y) e un vettore (v) di dimensione 100
Si definiscano in C il tipo struct e le variabili x,y,v, utilizzando le varie alternative possibili.

### Soluzione

Schema di dichiarazione 1: definizione del tipo `struct operazione`, definizione delle variabili utilizzando il tipo `struct operazione`

```
struct operazione
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
};
...
struct operazione x, y, v[100]
```

Schema di dichiarazione 2: dichiarazione contestuale di tipo e variabili

```
struct operazione
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
} x, y, v[100];
```

Schema di dichiarazione 3: dichiarazione contestuale di tipo e variabili, senza nome per il tipo struct

```
struct
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
} x, y, v[100];
```

 

Schema di dichiarazione 4: generazione del tipo `operazione_t`, mediante `typedef`

```
typedef struct operazione
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
} operazione_t;
...
operazione_t x, y, v[100];
```

Schema di dichiarazione 4-bis: generazione del tipo `operazione_t`, mediante `typedef`

```
struct operazione
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
};
...
typedef struct operazione operazione_t;
...
operazione_t x, y, v[100];
```

## 2 Esercizio

Siano date le seguenti dichiarazioni di tipi e variabili struct (con eventuale inizializzazione). Si individuino gli eventuali errori, proponendo le relative correzioni.

```
struct operazione
{
   int codice = 201;
   char nome[20] = “video al plasma”;
   float prezzo = 634.5;
   int npezzi = 7;
} op;
...
struct
{
   int x,
   int y,
   int x;
} a, b;
...
struct volo
{
   char partenza[20],
        arrivo[20];
   float durata
};
```

 

### Soluzione

I commenti indicano le parti errate e/o le correzioni.

```
/*
Il tipo di inizializzazione proposta (nella definizione di struct)
non e' legale. E' invece possibile inizializzare esplicitamente
la variabile op, come riportato nel seguito.
*/
struct operazione
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
} op = {201, "video al plasma", 634.5, 7};
...
/*
la dichiarazione multipla di campi dello stesso tipo non e'
corretta. Sono possibili due schemi alternativi: dichiarazioni separate,
terminanti ognuna con ';', oppure variabili separate da virgole, con tipo di
dati riportato una volta sola (vedi sotto)
*/
struct
{
   int x,
       y,
       x;
} a, b;
...
/*
la dichiarazione multipla e' corretta. Manca un ';' prima di '}'
*/
struct volo
{
   char partenza[20],
        arrivo[20];
   float durata;
};
```

## 3 Esercizio

Si vuole realizzare un tipo struct adatto a contenere informazioni anagrafiche di persone (cognome, nome, data di nascita, indirizzo di residenza). In particolare, i campi relativi a cognome e nome possono essere realizzati come stringhe di lunghezza inferiore a 30 caratteri, mentre data di nascita e indirizzo di residenza sono sotto-strutture, caratterizzate, a loro volta, da campi:

- Data: giorno, mese, anno (valori interi)
- Indirizzo: via (stringa), numero civico (intero), codice postale (stringa di 5 caratteri), città (stringa di meno di 30 caratteri), provincia (due caratteri).

Si definiscano in C i tipi struct adatti a rappresentare tali dati (si realizzino i 3 tipi definendoli mediante costrutto typedef).

 

### Soluzione

```
#define MAXS 30
#define NCAP 6
#define NPROV 3

typedef struct
{
  int g, m, a;
} data_t;

typedef struct
{
  char via[MAXS];
  int numero;
  char cap[NCAP], citta[MAXS}, prov[NPROV];
} indirizzo_t;

typedef struct
{
  char cognome[MAXS], nome[MAXS];
  data_t dataDiNascita;
  indirizzo_t indirizzo;
} persona_t;
```

## 4 Esercizio

Si ridefinisca la struttura (a due livelli) dell'esercizio precedente, utilizzando un'unica definizione (senza, cioè, definire separatamente i tipi relativi a data e indirizzo).

### Soluzione

```
#define MAXS 30
#define NCAP 6
#define NPROV 3

typedef struct
{
  char cognome[MAXS], nome[MAXS];
  struct {
    int g, m, a;
  } dataDiNascita;
  struct {
    char via[MAXS];
    int numero;
    char cap[NCAP], citta[MAXS}, prov[NPROV];
  } indirizzo;
} persona_t;
```

# 5 Esercizio

Sia dato il tipo struct dell'esercizio 3 (oppure 4). Si scrivano due funzioni leggiPersona e scriviPersona, rispettivamente per acquisire da tastiera e stampare a video i dati relativi a una struct. Per l'input, si prevede che vengano stampati come prompt i nomi di ognuno dei campi per i quali si fa input. Analogamente per l'output.

### Soluzione

```
persona_t leggiPersona (void)
{
  persona_t p;
  printf("cognome: "); scanf("%s", p.cognome);
  printf("nome: "); scanf("%s", p.nome);
  printf("data di nascita (g m a): ");
  scanf("%d%d%d", &p.dataDiNascita.g,
                  &p.dataDiNascita.m, &p.dataDiNascita.a);
  printf("indirizzo - via (sono ammessi spazi): ");
  gets(p.indirizzo.via);
  printf("indirizzo - numero: ");
  scanf("%d", &p.indirizzo.via);
  printf("indirizzo - numero: ");
  scanf("%d", &p.indirizzo.via);
  printf("indirizzo - cap: ");
  scanf("%s", &p.indirizzo.cap);
  printf("indirizzo – citta' (sono ammessi spazi): ");
  gets(p.indirizzo.citta);
  printf("indirizzo - provincia: ");
  scanf("%s", &p.indirizzo.prov);
  return p;
}
```

# 6 Esercizio

Si scriva una funzione in grado di contare il numero di giorni intercorsi tra due date, ricevute come parametri in ingresso. Si faccia riferimento ad un tipo data_t, realizzato mediante struct, quale quello utilizzato nell'esercizio 3.

### Soluzione

La funzione realizza il conteggio valutando la differenza tra anni, mesi e giorni.

```
int contaGiorni (data_t d0, data_t d1)
{
  int i, n, nM;

  /* conta anni */
  nM = d1.a – d0.a;
  n = nM*365;
  /* conta anni bisestili */
  for (i=d0.a+1; i<d1.a; i++) {
    if (i%4==0) n++;
  }

}
```