Strutture e Unioni

DD cap.10 pp.379-391, 405-406

KP cap. 9 pp.361-379

Strutture

- Strutture
  - Collezioni di variabili correlate (aggregati) sotto un unico nome
    - Possono contenere variabili con diversi nomi e tipi
  - Sono usate comunemente per definire i record da memnorizzare nei files
  - Combinate con i puntatori danno luogo a liste lincate, pile, code ed alberi

Definizione di strutture (1)

- Esempio:
```
struct card
{
  int valore;
  char seme;
};
```
  - La parola chiave `struct` introduce la definizione della struttura
  - `card` è il nome della struttura ed è usato per dichiarare variabili di quel *tipo di struttura*
  - `card` contiene due *membri* della struttura: un intero che contiene il valore della carta, ed un carattere che contiene l'iniziale del suo seme ('c', 'q', 'f', 'p')
  - I membri della stessa struttura devono avere nomi diversi, mentre membri di diverse strutture possono avere lo stesso nome (poi capiremo perché).
  - Ogni definizione di struttura deve terminare con `;`

Definizione di strutture (2)

  - La definizione di una struttura non riserva spazio in memoria, ma definisce un nuovo tipo composto, del quale si potranno definire variabili: `struct card` può essere usato come tipo.
  - Una struttura non può contenere un'istanza di se stessa ma può contenere un puntatore ad una struttura del suo stesso tipo:
- Esempio:
```
struct dipendente
{
  char nome[20];
  int eta;
  struct dipendente superiore; /*NO*/
  struct dipendente *ptr; /*ok*/
};
```

Definizione di strutture (3)

- Si osservi la dichiarazione:

```
struct card carta, mazzo[ 52 ], *cPtr;
```

- Si possono definire le stesse variabili anche in fase di definizione della struttura:

```
struct card
  {
    int valore;
    char seme;
  } carta, mazzo[ 52 ], *cPtr;
```

- Il nome della struttura è opzionale, ma in tal caso le variabili di quel tipo di struttura possono essere dichiarate solo contestualmente alla struttura
- Buona abitudine: usare sempre il nome delle strutture scegliendo nomi significativi.

Definizione di strutture (4)

Le uniche operazioni valide sulle strutture sono:

- Assegnare una struttura ad un'altra struttura dello stesso tipo
- rilevare l'indirizzo (&) di una variabile di tipo struttura
- accedere ai membri di una variabile di tipo struttura
- usare l'operatore sizeof per determinare la dimensione di una struttuta
- N.B. le strutture non si possono confrontare con == e !=

Inizializzare le strutture

- Le strutture possono essere inizializzate, allo stesso modo dei vettori, usando degli elenchi di inizializzazione
  Esempio: `struct card carta={3, 'c'};`
- Nel caso in cui nell'elenco ci fossero meno valori dei membri della struttura, si completa con 0 o NULL.
- L'inizializzazione può avvenire anche copiando una struttura su un'altra:
  Esempio:
  ```
  struct card carta={3, 'c'};
  altracarta=carta;
  ```

Accedere ai membri delle strutture

- Si accede ai membri di una strutture tramite:
  - operatore membro di struttura o operatore punto (.)
    Esempio:
    ```
    card miaCarta;
    printf( "%c", miaCarta.seme);
    ```
  - operatore puntatore a struttura o operatore freccia (->)
    Esempio:
    ```
    card *cPtr=&miaCarta;
    printf("%c", cPtr->seme);
    ```
  - Attenzione: -> va scritto senza spazi!
  - Attenzione: le seguenti 3 scritture sono equivalenti:
    `miaCarta.seme` `cPtr->seme` `(*cPtr).seme`
  - N.B. Se due strutture (ad es. card e plant) hanno un membro con lo stesso nome (seme), questo non è un problema:
    ```
    card miaCarta;  plant miaPianta;
    miaCarta.seme        miaPianta.seme
    ```
    non genera ambiguità!

Usare le strutture con le funzioni

- Le strutture possono essere passate alle funzioni:
  - Passando l'intera struttura
  - Passando singoli membri

  In entrambi i modi il passaggio è per valore
- Per passare le strutture per riferimento:
  - Passare l'indirizzo
- Vettori:
  - un vettore di strutture è passato per riferimento (è un vettore!)
  - creando una struttura contenente solo un vettore e passandolo si ottiene il passaggio di un vettore per valore

Esempio

Problema: dato un vettore di carte, dire se tra esse c'è una carta data.

```
struct card
{
  int valore;
  char seme;
} carta, mazzo[20];
```

```
void cerca(struct card x, struct card v[])
 {
  int i=0;
  while ((i<20) &&
            !((v[i].valore==x.valore)
        && (v[i].seme==x.seme)))
     i++;
  if (i==20)
    printf("carta non trovata")
  else
     printf("carta trovata");
 }
```

Un altro esempio

Problema: dato un elenco di studenti (memorizzati in un vettore) ciascuno con il proprio voto, contare il numero di studenti che ha preso 30.

```
struct studente
{
  char *nome;
  int matricola;
  int voto;
}class[20];
```

```
int conta30(struct studente a[])
 {
  int i, cont=0;
  for (i=0; i<20; i++)
     cont+=a[i].voto==30;
  return cont;
}
```

equivalentemente:
struct studente *a

Ancora un esempio

```
struct min_max CalcolamM(int v[], int n)
{
  struct min_max temp;
  if (n==1)
  {
    temp.min=v[0];
    temp.max=v[0];
    return temp;
  }
  else
  {
    temp=CalcolamM(v, n-1);
    if (v[n-1]<temp.min)
      temp.min=v[n-1];
    if (v[n-1]>temp.max)
      temp.max=v[n-1];
   return temp;
  }
}
```

Problema: si scriva una funzione che, dato un vettore, restituisca il minimo e il massimo del vettore.

Attenzione: le funzioni restituiscono solo un valore, a meno che…

```
struct min_max
{
  int min;
  int max;
};
```

typedef

- La parola chiave typedef crea dei nomi fittizi per i tipi di dato già definiti in precedenza.
  Esempio: typedef struct card Card;
  typedef Card * CardPtr;
- typedef non crea dei nuovi tipi, ma solo degli alias per quelli già esistenti
- Per abbreviare i nomi di struttura questi sono spesso definiti con typedef
  Esempio:
```
typedef struct
{
  int valore;
  char seme;
} Card;
```
- Buona abitudine: scrivere in maiuscolo la prima lettera dei tipi definiti con typedef

Le unioni (1)

- Un'unione, come una struttura, è un tipo di dato derivato i cui membri condividono lo stesso spazio in memoria.
- In varie situazioni, un programma può avere delle variabili utili in quel momento ed altre inutili: queste possono condividere lo spazio in memoria invece di sprecare.
- I membri di un'unione possono essere di qualsiasi tipo.
- Lo spazio occupato da un'unione è pari allo spazio occupato dal tipo del suo membro più grande.
- Le unioni contengono più tipi, ma si può far riferimento ad uno solo per volta.

Le unioni (2)

Esempio:
```
union number
{
  int x;
  double y;
};
```

- number è un tipo union e i suoi membri sono un intero e un double.
- La union crea solo un nuovo tipo di dato e non una variabile.
- Le operazioni che si possono eseguire su un'unione sono:
  - assegnare un'unione ad un'altra
  - rilevare l'indirizzo di una variabile di tipo unione
  - accedere ai membri di una unione usando gli operatori . e ->
  - non si possono usare == e !=.

Esempio

```
#include <stdio.h>
  union number{
    int x;
    double y;
  };
  int main()
  {union number val;
   value.x=100;
   printf("%d %f\n",value.x,value.y);
   value.y=100.0;
  printf("%d %f",value.x,value.y);
  return 0;
  }
```

Cosa stampa?
100 -9255…..
0 100.000000

Costanti di enumerazione

Il C fornisce il dato definito dall'utente detto enumerazione (enum), definito da un insieme di costanti intere rappresentate da identificatori. I valori di una enum iniziano da 0 e sono incrementati di 1.

Esempio:

```
enum mesi {GEN, FEB, MAR, APR, MAG, GIU,
LUG, AGO, SET, OTT, NOV, DIC};
```

enum crea un nuovo tipo i cui identificatori corrispondono agli interi da 0 a 11.

Per partire da 1:

```
enum mesi {GEN=1, FEB, MAR, APR, MAG, GIU,
LUG, AGO, SET, OTT, NOV, DIC};
```

Esempio

```
#include <stdio.h>
  enum mese{GEN=1, FEB, ...};

  int main()
  {enum mese m;
   char *nomemese[]={"", "gennaio", ...};
   for (m=GEN; m<=DIC; m++)
     printf("%d %s", m, nomemese[m]);
  return 0;
  }
```

può contenere un intero tra 1 e 12

vettore bidimensionale