Laboratorio di informatica

Ingegneria meccanica

Lezione 11 – 17 dicembre 2007

Record

- *Array*: collezione di dati *omogenei* (si accede al singolo dato specificando uno o più indici interi)
- Linguaggi di alto livello offrono tipicamente la possibilità di definire collezioni di dati *eterogenei*
- Una collezione di dati eterogenei (spesso indicata con il termine record) è caratterizzata dalla sua struttura in termini di numero e tipo degli elementi contenuti
- La descrizione di una collezione di dati eterogenei avviene specificando un certo numero di *campi*, ognuno dedicato a uno specifico *tipo di dato* e individuato da una stringa detta *nome del campo* (o *chiave*)

Strutture in C (1)

- In C, si può definire un tipo *collezione eterogenea* tramite il tipo derivato struct (struttura)
- Formato definizione del tipo *collezione eterogenea*
  struct nome-struttura { lista-membri } ;
  - nome-struttura: stesse regole date per il nome di una variabile
  - lista-membri: elenco di coppie terminate con ;, ognuna formata da un tipo-membro (un tipo base o derivato) e da un nome-membro (stesse regole usate per nome variabile, opportunamente completato nel caso di array, puntatori, etc.)
- Attenzione: per identificare il tipo così definito si scrive
  struct nome-struttura

Strutture in C (2)

- Esempio: definizione del tipo struct motore con i membri produttore (array di 20 char), annoproduzione (int), volume (int) e codicemagazzino (array di 100 char)

```
struct motore {
  char  produttore[ 20 ] ;
  int  annoproduzione ;
  float  volume ;
  char  codicemagazzino[ 100 ] ;
} ;
```

Nella definizione di un tipo struct possono comparire uno o più membri di altri tipi struct (può anche comparire un membro di tipo puntatore allo stesso tipo struct che si sta definendo: *strutture ricorsive*)

Strutture in C (3)

- Esempio di tipo struct con un membro di un tipo struct

```
struct produttore {
  char  codiceinternaz[ 100 ] ;
  char  dataultimoacq[ 20 ] ;
  int  codicequalita ;
} ;
struct componente {
  struct produttore origine ;
  int annoprod ;
  char codiceinterno[ 100 ] ;
} ;
```

- Tra i membri del tipo struct componente compare origine, di tipo struct produttore
- Il tipo struct produttore è definito prima del tipo struct componente

Strutture in C (4)

- L'uso di una variabile di un tipo struct definito richiede una preventiva dichiarazione, dove si specifica il tipo struct di interesse
- Si possono dichiarare variabili di un tipo struct, array di un tipo struct, puntatori a un tipo struct
  - Esempio con *variabile struttura*
    struct motore parte ;
    dichiara la variabile parte di tipo struct motore (motore è detto tipo struttura, parte è detta variabile struttura o struttura)

Strutture in C (5)

  - Esempio con *array di strutture*
    struct motore elencomotori[10] ;
    dichiara l'array elencomotori di 10 elementi di tipo struct motore (elencomotori è detto array di strutture)
  - Esempio con *puntatore a struttura*
    struct motore *motorePtr ;
    dichiara il puntatore motorePtr per il tipo struct motore (motorePtr è detto puntatore a struttura)
- Si può usare operatore sizeof su un tipo struct o su una variabile di un tipo struct

Definizione e uso di tipo struct (1)

- In un programma C, la definizione di un tipo struct e il suo uso nelle dichiarazioni delle variabili di interesse possono avvenire secondo varie modalità
- Modello adottato *in questo corso*
  - La definizione di un tipo struct è fatta prima della definizione della funzione main e prima di ogni prototipo di funzione (il tipo struct così definito è utilizzabile in tutte le funzioni contenute nello stesso file sorgente in cui appare la sua definizione)
  - La dichiarazione di variabili di un tipo struct, array di un tipo struct, puntatori a un tipo struct avviene specificando, nella forma struct nome-struttura, il tipo struct di interesse

Definizione e uso di tipo struct (2)

- Esempio

```
...
struct gatto {   /* definizione del tipo struct gatto */
   int  anni ;
   char  tipo[ 20 ] ;
   char  colore[ 10 ] ;
} ;
...
int main( ) {
struct gatto silvestro ;    /* dichiarazione variabile
                               silvestro di tipo struct gatto */
...
return 0 ;
}
```


Assegnazione di valori a strutture (1)

- Ai membri di una variabile di tipo struttura possono essere assegnati valori al momento della dichiarazione con meccanismo analogo a quello visto per gli array (*inizializzazione a tempo di dichiarazione*)

  Esempio

  ```
  struct gatto silvestro = { 1 , "ignoto" , "nero" } ;
  ```

  ha l'effetto di assegnare i valori 1 , "ignoto" e "nero" ai membri anni, tipo e colore, rispettivamente
- Se si specificano meno valori di quelli richiesti si ottiene
  - valore 0 (o 0.0 o '\0') per singole variabili o elementi di array di tipo intero (o reale o carattere) non inizializzati
  - valore NULL per tutti i puntatori non inizializzati

Assegnazione di valori a strutture (2)

- Ai singoli membri di una variabile di tipo struttura possono essere assegnati valori dopo la dichiarazione della variabile accedendo ai membri di interesse tramite gli operatori punto (.) o freccia (->)
  - Operatore punto usato con variabili di tipo struttura specificando nome della variabile e membro a cui si vuole accedere nella forma

    nome-variabile.nome-membro
  - Operatore freccia usato con puntatori a variabile di tipo struttura specificando nome del puntatore e membro a cui si vuole accedere nella forma

    nome-puntatore->nome-membro

    che equivale a (*nome-puntatore).nome-membro

Assegnazione di valori a strutture (3)

- Esempi: si assume romeo di tipo struct gatto (già definito)
  - Assegnazione di valori ai singoli caratteri del membro colore

    ```
    ...
    romeo.colore[ 0 ] = 'n' ;
    romeo.colore[ 1 ] = 'e' ;
    romeo.colore[ 2 ] = 'r' ;
    romeo.colore[ 3 ] = 'o' ;
    romeo.colore[ 4 ] = '\0' ;
    ```
  - Acquisizione tramite scanf dei caratteri del membro tipo

    ```
    ...
    printf( "\nInserire tipo (max 19 caratteri utili)\n") ;
    scanf( "%s" , romeo.tipo ) ;
    ```

Assegnazione di valori a strutture (4)

- Esempi: si assume gattoPtr puntatore a struct gatto
  - Assegnazione di valori ai singoli caratteri del membro colore della struttura puntata da gattoPtr

```
...
gattoPtr->colore[ 0 ] = 'o' ;
gattoPtr->colore[ 1 ] = 'c' ;
gattoPtr->colore[ 2 ] = 'r' ;
gattoPtr->colore[ 3 ] = 'a' ;
gattoPtr->colore[ 4 ] = '\0' ;
```

  - Acquisizione tramite scanf dei caratteri del membro tipo

```
...
printf( "\nInserire tipo (max 19 caratteri utili)\n") ;
scanf( "%s" , gattoPtr->tipo ) ;
```


Assegnazione di valori a strutture (5)

- A tutti i membri—in blocco—di una variabile di tipo struttura possono essere assegnati, dopo la dichiarazione della variabile, i valori di un'altra variabile di tipo struttura (con membri di valore definito)
- Esempio con silvestro e romeo di tipo struct gatto (con i valori dei membri di silvestro già definiti (in qualche modo))

```
romeo = silvestro ;
```

  ha l'effetto di assegnare a tutti i membri della variabile destinazione (romeo) i valori che i membri corrispondenti hanno nella variabile sorgente (silvestro)
- Attenzione: Questa modalità di assegnazione non è consentita tra variabili di tipo array!

Precedenza e associatività per '.' e '->'

. ->	da sin a dx
! ++ -- + - (tipo) * & sizeof	da dx a sin
* / %	da sin a dx
+ -	da sin a dx
< <= > >=	da sin a dx
== !=	da sin a dx
&&	da sin a dx
||	da sin a dx
= += -= *= /= %=	*da dx a sin*

Strutture e funzioni (1)

- Si possono passare a una funzione singoli membri di una struttura: l'uso che la funzione può fare del valore del singolo membro dipende, come già visto, dalla natura di questo
  - se è un array, la funzione riceverà uno strumento con cui potrà accedere in lettura/scrittura ai suoi elementi
  - se è un puntatore, la funzione riceverà uno strumento con cui potrà accedere in lettura/scrittura alla variabile puntata da questo
  - se è un numero/carattere la funzione riceverà soltanto una copia del valore di questo

Strutture e funzioni (2)

- Si possono passare a una funzione puntatori a singoli membri di una struttura: la funzione riceverà uno strumento con cui potrà accedere in lettura/scrittura al singolo membro
- L'indirizzo di un singolo membro può essere ottenuto con l'operatore & (per membri array, & è omesso)
- Esempio

  Se peso è un membro (p.es., int) del tipo struct oggetto e tavolo è una variabile di tipo struct oggetto, allora &(tavolo.peso) può essere usato per puntare al membro peso della struttura tavolo

  &(tavolo.peso) *equivale* a &tavolo.peso (vedi tabella *precedenza e associatività*)

Strutture e funzioni (3)

- Si può passare a una funzione un puntatore alla struttura completa: la funzione riceverà uno strumento con cui potrà accedere in lettura/scrittura a ogni membro
- L'indirizzo di una struttura può essere determinato con l'operatore &
- Esempio

  Se tavolo è una variabile di tipo struct oggetto, allora &tavolo può essere usato per puntare alla struttura tavolo

Strutture e funzioni (4)

- Si può passare a una funzione una struttura completa: in questo caso l'uso che la funzione può fare del valore del singolo membro dipende dalla natura di questo
  - se è un puntatore, la funzione riceverà uno strumento con cui potrà accedere in lettura/scrittura alla variabile puntata da questo
  - se è un array, la funzione riceverà soltanto una copia del valore dei suoi elementi
  - se è un numero/carattere la funzione riceverà soltanto una copia del valore di questo

Strutture e funzioni (5)

- Si può specificare un tipo struct definito per il valore che una funzione deve restituire: una funzione può restituire una struttura
- Attenzione: Questa possibilità è esclusa nel caso di un array!
- Esempio: nelle slide seguenti si definiscono il tipo struct compl (per introdurre numeri complessi) ed esempi di funzioni per somma, inizializzazione e confronto di numeri complessi

Strutture e funzioni (6)

```
struct compl {
   double re ;
   double im ;
} ;
...
struct compl som( struct compl , struct compl ) ;
...
d = som( c1 , c2 ) ; /* c1 , c2 , d  di tipo struct compl */
...
struct compl som( struct compl a, struct compl b)
{
  struct compl c ;
  c.re = a.re + b.re ;
  c.im = a.im + b.im ;
  return c ;
}
...
```


Strutture e funzioni (7)

```
/* struct compl {
      double re ;
      double im ;
} ; */
...
void setcompl( struct compl * , double , double ) ;
...
setcompl( &d , c1 , c2 ) ; /* c1 , c2 di tipo double
                d  di tipo struct compl */
...
void setcompl( struct compl *a , double rr , double ii )
{
   a->re = rr ;
   a->im = ii ;
   return ;
}
...
```


Strutture e funzioni (8)

```
/* struct compl {
      double re ;
      double im ;
} ; */
...
int  uguali( struct compl , struct compl ) ;
...
if ( uguali ( a , b ) )  { ... }    /* a, b di tipo struct compl */
...
int uguali(struct compl c1, struct compl c2)
{
   if (c1.re == c2.re && c1.im == c2.im)
       return 1;
   else
      return 0;
}
```


Strutture e funzioni (9)

```
...
struct compl {
   double re ;
   double im ;
} ;
...
void setcompl( struct compl * , double , double ) ;
struct compl som( struct compl , struct compl ) ;
int  uguali( struct compl , struct compl ) ;
...
int main( ) {
   ...
   ...
   return 0 ;
}
...
/* definizioni funzioni */
...
```

Esercizio

```
struct data {
    unsigned short int giorno;
    unsigned short int mese;
    short int anno;
};
```

oppure

```
struct data {
    unsigned short int giorno;
    char mese[10];
    short int anno;
};
```

modificare esercizio "data giorno successivo" usando la struct (Esercitazione 3, esercizio 3)

Confronto di due stringhe. La funzione della libreria string.h

```
int strcmp(const char *s1, const char *s2);
```

restituisce 0 se le due stringhe sono uguali

Ordinamento di array

- **Ordinare** un array significa disporre i suoi elementi secondo un ordine specificato. Per esempio, gli elementi di un array di interi possono essere disposti dal più piccolo al più grande
- *Algoritmo elementare* di ordinamento di un array monodimensionale (per fissare le idee, consideriamo ordine crescente ed N elementi interi)

  IDEA: Cercare minimo elemento a partire dalla prima posizione e disporlo in prima posizione (scambio di due elementi, se necessario). Cercare il minimo elemento a partire dalla seconda posizione e disporlo in seconda posizione (scambio di due elementi, se necessario). Etc.

Ordinamento di array (2)

- Funzione utile:

  ricerca da indice k a indice N-1 dell'indice m dell'elemento minimo nell'array
- Ciclo con chiamata di ricerca con k variabile da 0 a N-2 e scambio condizionato degli elementi di indici k e m (condizione di scambio: m != k)
- Possibile effettuare il ciclo all'interno di una funzione ordina

C: Ricerca ind. val. minimo, da indice k

```
unsigned min1( int [ ] , unsigned , unsigned ) ;

unsigned min1( int a[ ] , unsigned k , unsigned N )
{
/* si assume N >= 1 , k coerente con N */
unsigned int h , m = k ;
int mini = a[ k ] ;

for ( h = k+1 ; h < N ; h++ )
      if ( a[ h ] < mini ) {
            mini = a[ h ] ;
            m = h ;
      }
return m ; /* m = 1a posizione valore minimo */
}
```

C: Funzione ordina

- `void ordina( int [ ] , unsigned ) ;`

- `void ordina( int a[ ] , unsigned N ) {`
- `/* si assume N >= 1 */`

```
int tmp ;
unsigned int hh , kk ;
for ( hh = 0 ; hh < N-1 ; hh++ ) {
     kk = min1 ( a , hh , N ) ;
    if ( kk != hh ) {
       tmp   = a[ kk ] ;
       a[ kk ]  = a[ hh ] ;
       a[ hh ]  = tmp ; }
 }
}
```


Ordinamento bubblesort (1)

- Algoritmo bubblesort ordina un array monodimensionale iterando un procedimento di scambio condizionato tra coppie di elementi successivi (N interi di indici 0, 1, .., N-1)
- In una iterazione, si esegue una scansione dell' array in cui ogni elemento viene confrontato con l'elemento successivo e, se necessario, i due elementi vengono scambiati di posizione. Una iterazione sposta gli elementi più *pesanti* (interi più grandi) verso gli indici più elevati
- La prima iterazione sposta in ultima posizione (N-1) il valore massimo nell'array (estremi scansione: da indice 0 a indice N-2)

Ordinamento bubblesort (2)

- ESEMPIO: Iterazione 1
  [7 , 4 , 1 , 5 , 3] originale
  [4 , 7 , 1 , 5 , 3]
  [4 , 1 , 7 , 5 , 3]
  [4 , 1 , 5 , 7 , 3]
  [4 , 1 , 5 , 3 , 7] risultato 1a iterazione

- La seconda iterazione sposta in penultima posizione (N-2) il valore massimo presente tra la prima e la penultima posizione dell'array prodotto dalla prima iterazione (estremi scansione: da indice 0 a indice N-3). Etc.

Ordinamento bubblesort (3)

- ESEMPIO:
  [7 , 4 , 1 , 5 , 3] originale
  [4 , 1 , 5 , 3 , 7] risultato 1a iterazione
  [1 , 4 , 3 , 5 , 7] risultato 2a iterazione
  [1 , 3 , 4 , 5 , 7] risultato 3a iterazione

- Sono necessarie al massimo N-1 iterazioni: si può interrompere l'algoritmo dopo la prima iterazione che non produce scambi di posizione tra elementi

Bubblesort in C (1)

```
for ( j = 1 ; j <= N-1 ; j++ )
    for ( i = 0 ; i < N-1 ; i++ )
            if ( v[ i ] > v[ i+1 ] ) {
                 temp = v[ i ] ;
                 v[ i ] = v[ i+1 ] ;
                 v[ i+1 ] = temp ;
               }
```

- Viene eseguito il massimo numero di iterazioni
- Ad ogni iterazione, si esegue la scansione completa dell'array

Bubblesort in C (2)

```
do { scambio = 0 ;
     for ( i = 0 ; i < N-1 ; i++ )
           if ( v[ i ] > v[ i+1 ] ) {
                 temp = v[ i ] ;
                 v[ i ] = v[ i+1 ] ;
                 v[ i+1 ] = temp ;
                 scambio = 1 ; }
   }
while ( scambio == 1 ) ;
```

- Viene eseguito il numero necessario di iterazioni
- Ad ogni iterazione, si esegue la scansione completa dell'array

Bubblesort in C (3)

```
n = N ;
do { scambio = 0 ;
     for ( i = 0 ; i < n-1 ; i++ )
           if ( v[ i ] > v[ i+1 ] ) {
                 temp = v[ i ] ;
                 v[ i ] = v[ i+1 ] ;
                 v[ i+1 ] = temp ;
                 scambio = 1 ; }
      n-- ;
    }
while ( scambio == 1 ) ;
```

- Eseguite iterazioni necessarie, ogni iterazione scansiona solo la porzione di array che è necessario esplorare

Ricerca Binaria (1)

- Algoritmo applicabile per la ricerca di un elemento in un array ordinato
- Ricerca di elemento in un array disordinato:
  - Elemento cercato può trovarsi o meno nell'array
  - Se presente, può apparire in una o più posizioni indipendenti (in caso di occorrenze multiple)
  - Necessaria una scansione completa dell'array
- Ricerca di elemento in un array ordinato:
  - Elemento cercato può trovarsi o meno nell'array
  - Se presente, può apparire in una o più posizioni contigue (in caso di occorrenze multiple)
  - Non è necessaria una scansione completa dell'array

Ricerca Binaria (2)

- Consideriamo array v di N interi in ordine *crescente* (v[i] < v[i+1], i = 0, 1, … N-2) e *cerchiamo* l'elemento val
- Idea base per la ricerca

  1.Valutare posizione rispetto al *centro* (*posizione centrale* m = (N-1)/2): se v[m] > val , cercare nella prima metà dell'array, se v[m] < val, cercare nella seconda metà dell'array, se v[m] = val, risposta cercata è m

  2.Ripartire cercando nella metà individuata (adattando indice minimo ed indice massimo per il nuovo passo di ricerca)

  3. Etc.

Ricerca Binaria (3)

- Nel caso in cui l'ordinamento sia *non decrescente* (v[i] <= v[i+1]), l'azione per il caso v[m] = val deve essere decisa in base alle specifiche esigenze
- Se non è sufficiente rispondere che l'elemento è presente (p.es., se si vogliono tutte le posizioni di occorrenza), l'azione da svolgere una volta trovato il valore cercato richiederà la valutazione (di alcuni) degli elementi nelle posizioni prossime ad m
- ESEMPIO: Cerco la *minima posizione di occorrenza* di 7 in [0, 0, 2, 3, 4, 7, 7, 8, 9, 9, 9]. Trovato 7 in posizione 5, devo controllare se ci sono altri 7 nelle posizioni precedenti la 5

Esempio di Ricerca Binaria in C (1)

```
int b2search( int v[ ] , int size , int val ) {
/* -1 se val assente in v o la posizione di occorrenza
   di val in v (ordinamento strettamente crescente) */
   int min, max, m ;
   min = 0 ; max = size-1 ;
   while( min <= max ) {
     m = ( min + max )/2 ;
     if( val < v[ m ] ) max = m-1 ;
     else if( val > v[ m ] ) min = m+1 ;
     else return m ;
    }
  return -1 ;
}
```


Esempio di Ricerca Binaria in C (2)

- Se necessario, una volta individuata un'occorrenza in posizione pval del valore val cercato nell'array ordinato, si può determinare se esistono occorrenze multiple
- Array ordinato → eventuali occorrenze multiple di val si trovano in posizioni successive (pval è una di queste)
- Ricerca della successione di occorrenze può essere eseguita da una funzione

Esempio di Ricerca Binaria in C (3)

```
int multiple( int v[ ] , int size , int val , int pos ,
              int *start ) {
/* restituisce numero totale N di occorrenze di
   val in v ( si assume v[ pos ] = val ) e lascia in
   *start la posizione della prima occorrenza */
   int p = pos-1 , cont = 0 ;
   while(( p >= 0 ) && ( v[ p ] == val )) {
     cont++ ; p-- ; }
   *start = pos-cont ; p = pos+1 ;
   while (( p <= size-1 ) && ( v[ p ] == val )) {
     cont++ ; p++ ; }
  return ++cont ;
}
```


Ricerca Binaria: Complessità

- Array ordinato di N elementi
  Numero di passi necessario per determinare se un dato valore è presente o no nell'array dipende dall'algoritmo di ricerca
- Ricerca binaria
  Ad ogni passo, la dimensione dell'array in cui si effettua la ricerca viene divisa per 2 (bisezione) → numero di passi massimo è circa log2(N)
- Ricerca lineare
  Partendo da un estremo, si esegue la scansione dell'array fino al primo incontro del valore cercato → numero di passi massimo è N

Generazione di numeri casuali (1)

- La funzione rand genera sequenze di numeri pseudo-casuali, con valori compresi tra 0 e la costante RAND_MAX (definita in stdlib.h)
- La sequenza di numeri generata da rand si ripete identica ogni volta che il programma viene eseguito
- Per produrre sequenze diverse si deve variare il seme usato per inizializzare il generatore di numeri pseudo-casuali con la funzione srand
- La funzione srand riceve un argomento di tipo unsigned (il seme) che condiziona la sequenza generata da rand (srand deve essere chiamata una volta sola all'interno del programma)
- Per generare il seme, tipicamente viene sfruttata la funzione time della libreria time.h che, invocata con parametro 0 restituisce un valore unsigned che rappresenta l'ora corrente del giorno espressa in secondi.
- In questo modo srand viene chiamata ogni volta con un valore diverso
- Tipica chiamata di srand:
  `srand ( time(0) );`

Generazione di numeri casuali (2)

- Spesso si vogliono generare numeri casuali in intervalli definiti, a volte in forma razionale e non intera.

Esempi:

numeri casuali interi tra 0 e N-1

```
varint = rand ( ) % N        /* varint di tipo int */
```

numeri casuali tra 1 e N

```
varint = rand ( ) % N +1     /* varint di tipo int */
```

numeri casuali razionali tra 0 e 1

```
vardouble = (double) rand ( ) / RAND_MAX
                             /* vardouble di tipo double */
```

numeri casuali razionali tra 0 e N

```
vardouble = (double) rand ( ) /RAND_MAX * N
                             /* vardouble di tipo double */
```

Generazione di numeri casuali (3)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define DIM 10

int main ( )
{
  unsigned seed, i;
  double a[DIM];

  seed = time (0);
  srand (seed);        /* oppure srand ( time (0) ); */

  for ( i = 0 ; i < DIM ; i++ )
     a[ i ] = ( double ) rand ( )/ RAND_MAX  * 1000;

  for ( i = 0 ; i < DIM ; i++ )
    printf ( "Indice= %u, valore= %lf\n" , i , a[i]);
  return 0;
}
```