Laboratorio di informatica

Ingegneria meccanica

Lezione 9 – 3 dicembre 2007

Puntatori in C (1)

- Un puntatore è una variabile di tipo speciale, il cui valore permette di accedere (*puntare*) ad una variabile (detta *variabile puntata*)
- Se il valore del puntatore p è utilizzabile per accedere alla variabile v, si dice che v è puntata da p (o che p punta a v)
- Una variabile di tipo puntatore contiene l'indirizzo della variabile a cui punta
- In C, il tipo puntatore è un *tipo derivato* (anche l'array, mentre gli altri sono tipi base).
- Formato dichiarazione: tipo-puntato *nome ;
  ESEMPIO: int *iPtr ;
- iPtr è così dichiarato per puntare a una variabile di tipo int (attenzione: il valore di iPtr è per ora indefinito)

Puntatori in C (2)

- Il simbolo * applicato ai puntatori indica un operatore unario detto " operatore di risoluzione del riferimento" (indirezione)
- Applicando l'operatore * ad un puntatore si accede alla variabile puntata da questo (per acquisirne o modificarne il valore)
- Il simbolo & (operatore indirizzo) applicato a una variabile indica l'indirizzo di memoria della variabile
- E' possibile far puntare un puntatore ad una particolare variabile usando l'operatore indirizzo & (attenzione: & è applicabile solo a variabili (non applicabile a costanti, espressioni, (e altro!))
- Se p punta a v è possibile accedere al valore di v (per acquisirlo o per modificarlo) usando sia l'espressione v sia l'espressione *p

Puntatori in C (3)

ESEMPIO

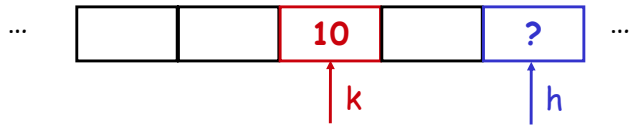
```
int *iPtr ;       /* iPtr può puntare ad una var. int */
int  ia=4, ib, ic=8 ;  /* ia, ib, ic variabili int */
iPtr = &ia ;      /* iPtr punta alla variabile ia */
ib = *iPtr;    /* ib assume valore di ia (4) */
 *iPtr = ic   ;    /* ia assume valore di ic (8) */
```

Puntatori in C (4)

- Significato di "int *iPtr ;": risolvendo il riferimento di iPtr, si accede ad una variabile di tipo int
- Attenzione: puntatore dichiarato per tipo specificato
- Attenzione: operatore indirizzo & applicabile solo a variabili (no a costanti, no ad espressioni)
- Possibile utilizzare la variabile puntata da un puntatore in ogni espressione compatibile
  ESEMPI /* int *i , *j , x , y ; */
  *i = *j + 8 ; y = x + *j ; *i = *i + 22 ; *i += *j ;
- Possibile assegnare il valore di un puntatore ad un altro puntatore compatibile
  ESEMPIO /* int *i , *j ; */
  i = j ; /* i e j puntano ora alla stessa variabile */

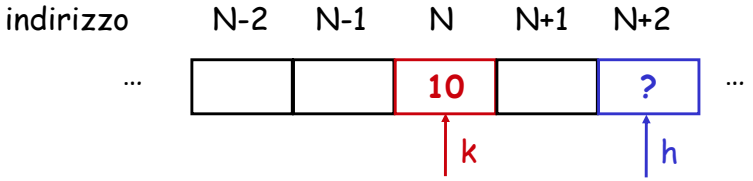
Variabili (1)

- Il nome di una variabile (di tipo qualunque) è univocamente associato alla locazione di memoria in cui il valore della variabile è registrato
- ESEMPIO
  int k = 10 , h ; /* valore di h non definito */



- L'accesso alla locazione assegnata ad una variabile (cioè *l'accesso al valore di una variabile*) avviene usando il *nome* della variabile (*il nome corrisponde all'indirizzo della locazione assegnata*)

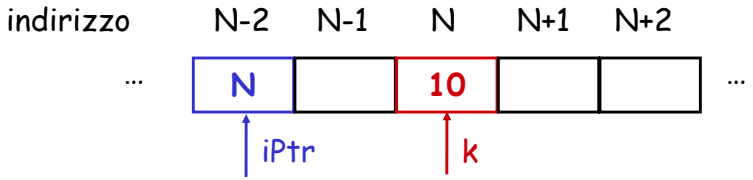
Variabili (2)



- *In questa rappresentazione: 1. Si assume che una locazione di memoria abbia capacità sufficiente per la memorizzazione di un valore di tipo qualunque; 2. Le posizioni di k ed h in memoria sono arbitrarie*

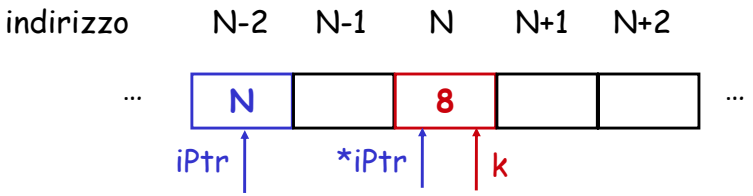
Variabili (3)

- Quanto detto vale anche per variabili di tipo puntatore
- ESEMPIO
  int k = 10 , *iPtr ;
  iPtr = &k ; /* valore di iPtr è l'indirizzo associato a k */



Variabili (4)

```
int k = 10 , *iPtr ;
iPtr = &k ;  /* iPtr punta a k */
*iPtr = 8 ;
```

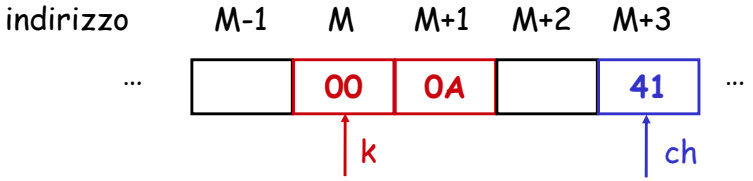


- L'accesso al valore di k tramite iPtr si ottiene applicando ad iPtr l'operatore * di risoluzione del riferimento (*iPtr *attualmente* associato alla stessa locazione a cui è associato k)

Dimensioni delle variabili (1)

- Il numero di byte impiegati per memorizzare il valore di una variabile/costante dipende dal suo tipo (già visti vari esempi)
- In C, ogni byte è univocamente individuabile da un indirizzo (unità di memoria indirizzabile è il byte)
- L'indirizzo associato ad una variabile/costante è l'indirizzo del primo byte in memoria in cui il valore è registrato
- Il nome di una variabile è associato all'indirizzo del primo byte in memoria in cui il suo valore è registrato (l'accesso è però all'intero valore!)

Dimensioni delle variabili (2)



- 1. Una locazione di memoria ha capacità di un byte;
- 2. Le posizioni di k ed ch in memoria sono arbitrarie;
- 3. I valori di k (10) ed ch (65 ('A' in ASCII)) sono rappresentati in esadecimale;
- 4. k occupa due byte (un tipo intero) ed h occupa un byte (tipo char)

C: Funzione sizeof (1)

- **La dimensione in byte di una variabile (array incluso) o di un tipo di dato può essere fornita dalla funzione sizeof**
- **sizeof è valutata *durante la compilazione* e restituisce il numero di byte impiegati per l'argomento richiesto**
- **sizeof restituisce un valore di tipo size_t (definito in stddef.h): assimilabile ad un intero senza segno (può essere necessario l'uso dell'operatore cast per forzare la conversione al tipo intero desiderato)**
- **Uso: sizeof variabile/array o sizeof(tipo)**

C: Funzione sizeof (2)

- ESEMPI

  printf("%d", (int) sizeof(double)) ;

  stampa il numero di byte impiegati per un double

  printf("%d", (int) sizeof k) ;

  stampa il numero di byte impiegati per la variabile k

  printf("%d", (int) sizeof(int *)) ;

  stampa il numero di byte impiegati per un puntatore al tipo int

  printf("%d", (int) sizeof(double *)) ;

  stampa il numero di byte impiegati per un puntatore al tipo double

C: Funzione sizeof (3)

- Si può usare sizeof con array di dimensione definita
- ESEMPIO

  double v[10] ;

  printf("sizeof v = %d ", (int) sizeof v) ;

  stampa il numero di byte impiegati per 10 double
- Il numero di elementi di un array di dimensione definita può essere ottenuto dividendo il valore di sizeof nome-array per il valore di sizeof(tipo-elementiarray)
- ESEMPIO (continua da sopra)

  ((int) sizeof v) / ((int) sizeof(double))

C: Puntatori in C (4)

- Ad un puntatore (ad un tipo qualunque) può essere assegnato un valore speciale NULL per indicare la situazione in cui il puntatore non punta ad alcuna variabile
- Può essere usato come valore di inizializzazione di un puntatore
- ESEMPIO
- int *iPtr ; /* iPtr di valore indefinito */
- int *jPtr = NULL ; /* jPtr di valore definito ma non utilizzabile per l'accesso a variabili */
- La specifica di conversione per la stampa di puntatori è %p

C: Puntatori in C (5)

- Su un puntatore possono essere eseguite le operazioni di
  - Incremento e decremento (++/--)
  - Somma di un intero
  - Sottrazione di un intero
- Se si somma/sottrae un intero ad/da un puntatore, questo viene incrementato/decrementato dell'intero moltiplicato per la dimensione in byte del tipo a cui il puntatore punta
- Se iPtr è un puntatore ad un intero, ed un intero occupa 4 byte, iPtr += 2 ; incrementerà il valore di iPtr di 8 (*scansione di un array con puntatore*)

Precedenza e associatività

! ++ -- + - (tipo) * &	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

Puntatori in C (6)

- **Attenzione** all'uso di pre/post incremento e pre/post decremento con puntatori (guardare la tabella "precedenza e associatività di operatori")
  ESEMPI
  /* int *i , *j ; */
  ++*i ; equivale a ++(*i) ; variabile puntata sarà pre-incrementata
  *i++ ; equivale a *(i++) ; variabile puntatore sarà post-incrementata (espressione i++ valida perché ++ è un operatore valido per puntatori (vedi *aritmetica dei puntatori*))

Puntatori: esercizi (1)

```
int a;
double b, *pd;
pd = &a;
```
[Warning] assignment from incompatible pointer type

```
double b, c, *pd;
b = 3.14;
c = &(*pd);
```
[error] incompatible types in assignment

```
double b, *pd;
pd = &b;
*pd = 3.14;
```
ok

```
double b, c, *pd;
b = 3.14;
pd = &c;
*pd = &b;
```
[error] incompatible types in assignment

Puntatori: esercizi (2)

```
int a;
double b = 3.14, *pd;
pd = &b;
a = *pd;
```
ok, attenzione troncamento

```
double b, c, *pd;
b = 3.14;
pd = &c;
*pd = b;
```
ok

```
double b, *pd;
*pd = 3.14;
b = *pd;
```
compilazione ok

Puntatori: esercizi (3)

```
double b, *pd;
*pd = 3.14;
&b = pd;
```

[error] invalid lvalue in assignment

```
int a;
a = &2;
```

[error] invalid lvalue in unary `&'

```
double b, *pd;
*pd = *&b;
```

Compilazione ok

C: Puntatori e funzioni

- Possibile definire funzioni che hanno puntatori tra i propri argomenti
- Nella chiamata di una funzione, gli argomenti attuali sono sempre passati per valore
- Una funzione lavora su copie locali degli argomenti passati
- Quando si passa un puntatore, si fornisce alla funzione uno strumento per accedere alla variabile puntata dal puntatore
- Passare un array ad una funzione equivale a passare un puntatore al primo elemento dell'array

C: Funzione con argomenti puntatori

```
void scambia( int *iPtr , int *jPtr ) ;

void scambia( int *iPtr , int *jPtr ) {
  int temp ;
  temp = *iPtr ;  /* vengono scambiati i valori
   *iPtr = *jPtr ;     delle variabili puntate da
   *jPtr = temp ;    da iPtr e jPtr */
}


int h , k ;
…
scambia( &h , &k ) ;
```


C: Funzione senza argomenti puntatori

```
void scambia( int i , int j ) ;

void scambia( int i , int j ) {
  int temp ;
  temp = i ;
   i = j ;
   j = temp ;
}


int h , k ;
…
scambia( h , k ) ;      /* valori delle variabili h e k
                           immutati */
```

Esempio: min e max di un array

```
void mM( float [ ] , float * , float * , int );

void mM( float v[ ] , float *pm , float *pM , int dim ) {
  int i ;
  *pm = *pM = v[ 0 ] ;
  for ( i = 1 ; i < dim ; i++ ) {
      if ( v[i] < *pm )    *pm = v[i] ;
      if ( v[i] > *pM )    *pM = v[i] ;
   }
}


 float vint[N], min, max ;
 ...
 mM( vint , &min, &max, N ) ;
```