Laboratorio di informatica

Ingegneria meccanica

Lezione 8 – 26 novembre 2007

Rappresentazione di numeri reali (1)

- Numerazione posizionale: $d_n \cdots d_1 d_0 \,.\, d_{-1} d_{-2} \cdots d_{-m}$ rappresenta
  $d_n B^n + \cdots + d_1 B^1 + d_0 + d_{-1} B^{-1} + \cdots + d_{-m} B^{-m}$
- Notazione scientifica: $S \times B^{Exp}$
  S è detto significando
  Exp è detto esponente
- Esempi
  $-7869000000000000000000000 \rightarrow -7.869 \times 10^{25}$
  $0.0000000000000000000001234 \rightarrow 1.234 \times 10^{-21}$
- Attenzione: $S \times B^{Exp}$ può presentarsi nelle forme SEExp o SeExp (-7.869E25, 1.234e-21)

Rappresentazione di numeri reali (2)

- Formati tipici nel caso binario: 32 o 64 bit, di cui
  1 bit di segno (0 positivo e 1 negativo)
  8 o 11 bit di esponente
  23 o 52 bit di significando (in valore assoluto)
- Il significando è posto nella forma normalizzata
  1.b ··· bb (rappresentazione di 1. è implicita) (il numero 0 è rappresentato in modo speciale)
- L'esponente Exp è *polarizzato*: Exp è rappresentato con Exp+bias (bias = 127 o 1023)

Rappresentazione di numeri reali (3)

- Esempio su 32 bit: $(-0.75)_{10}$ equivale a $(-0.11)_2$
  Segno: 1 (-)
  Normalizzazione: $0.11 \rightarrow 1.1 \times 2^{-1}$
  Significando: solo un 1 di peso -1, ovvero, su 23 bit, 10000000000000000000000 (1. implicito)
  Polarizzazione esponente: $-1 \rightarrow -1+127 = 126$ (esponente polarizzato = 01111110)
  $-0.75 \rightarrow$ 1 01111110 10000000000000000000000
- Limiti precisi per gli intervalli in cui sono *spalmati* i valori rappresentabili. In modulo, valori troppo piccoli provocano underflow, valori troppo grandi provocano overflow

C: Tipi float, double, long double

- Minimo intervallo, in modulo: [1E-37, 1E+37]
- Minima precisione in cifre decimali: 6 per float, 10 per double e long double
- Dettagli dell'implementazione in "float.h" (varie grandezze disponibili)
- Rappresentazione tipica: quella descritta per i reali (32 bit per float, 64 bit per double)
- Nelle dichiarazioni: float, double, long double
- Specifica di conversione: %f, %lf, %Lf (altre possibilità in seguito)

C: Operandi reali (1)

- Variabili di tipo float, double o long doble
- Costanti usate direttamente nelle espressioni

  Segno *opzionale*; cifre decimali; senza spazi vuoti

  Separazione parte intera-parte frazionaria con punto; *una delle due parti può mancare*

  Esponente: e o E (-2.67e-23; +0.098E+32); *opzionale*, se presente, è seguito da un intero

  Se manca il punto *deve* essere presente l'esponente, e viceversa (-267e-2; +0.098);

  Suffisso: f o F per float e l o L per long double; *opzionale* (-267.0000876787e29L)

C: Operandi reali (2)

- Costanti introdotte con #define

  Forma: #define nome valore

  Nome: stesse regole date per il nome di variabili

  Valore: per produrre valori *corretti*, stesse regole date per le costanti nelle espressioni
- Il *tipo* di una costante reale *dipende* da come viene specificata

  Senza suffisso: assegnato tipo double

  Con suffisso f o F: assegnato tipo float

  Con suffisso l o L: assegnato tipo long double

C: Operatori per reali (1)

- ARITMETICI

  Binari (due operandi): somma (+), sottrazione (-), prodotto (*), quoziente (/)

  Unari (un operando): segno (+, -)
- Attenzione: il comportamento in caso di underflow/overflow *dipende* dall'implementazione

C: Operatori per reali (2)

- **RELAZIONALI**

  Binari (due operandi): maggiore (>), minore (<), maggiore/uguale (>=), minore/uguale (<=), uguale (==), diverso (!=)
- **Attenzione**: la rappresentazione *approssimata* dei valori reali può produrre sorprese

  Esempio: (1.0/3.0 + 1.0/3.0 + 1.0/3.0) == 1.0 potrebbe valere 0

  Test del tipo r1 uguale r2 tipicamente sostituito con test modulo(r1-r2) minore/uguale D, con D opportuno

C: Operatori per reali (3)

- **LOGICI**

  Binari (due operandi): AND (&&), OR (||)

  Unario (un operando): NOT (!)
- Vengono applicate le tabelline viste per gli interi (per la valutazione, un operando può essere solo 0.0 o diverso da 0.0)

  Esempi: !-234.8 = 0 ; +1.1 && -1.1 = 1 ; 0.1 || 0.0 = 1
- Ordine di valutazione operandi per && e ||: stesse regole viste per gli interi

C: Libreria matematica

- Collezione di funzioni matematiche predefinite, utilizzabili inserendo la direttiva

  # include <math.h>
- **Esempi**

  double sqrt(double x): radice quadrata non negativa di x

  double pow(double x , double y): x elevato alla potenza y

  double floor(double x): *intero* più grande non maggiore di x

  double ceil(double x): *intero* più piccolo non minore di x

C: Conversione di tipo (1)

- Conversione *implicita* nel caso di operatori binari utilizzati con due operandi di tipo diverso: uno dei due tipi (inferiore) viene promosso all'altro (superiore) ed il risultato è del tipo superiore
- ESEMPIO di regola (informale)

  oper1 * oper2: se oper1 è double e oper2 è float, oper2 *promosso* a double e risultato è double
- **Attenzione:** a , b float e c double

  a = b + c ; assegnerà il valore double risultato della somma ad una variabile float con potenziale *perdita di informazione* (in un assegnamento il tipo della variabile destinazione definisce il tipo finale)

C: Conversione di tipo (2)

- Conversione *implicita* nella chiamata di funzione
- Gli argomenti passati ad una funzione *dichiarata con prototipo* vengono convertiti secondo quanto specificato nella dichiarazione con potenziale *perdita di informazione*
- ESEMPIO: se gli argomenti formali sono float e int, una chiamata con argomenti attuali int e float provoca una potenziale *perdita di informazione*

C: Conversione di tipo (3)

- Conversione *implicita* nella return all'interno di una funzione
- In return (espressione), il tipo assegnato al valore di espressione è quello specificato nella dichiarazione della funzione (potenziale *perdita di informazione*)
- ESEMPIO: b float e c int, tipo di ritorno int return b * c ; convertirà il risultato float del prodotto in un int con potenziale *perdita di informazione*

C: Conversione di tipo (4)

- Conversione *esplicita* di tipo può essere forzata con operatore cast
- (tipo) espressione provoca la valutazione di espressione come se il risultato dovesse essere assegnato ad una variabile del tipo forzato
- ESEMPIO di utilizzazione: conversione tipo argomento nella chiamata funzioni di libreria (potrebbero non fare uso di dichiarazione tramite prototipo)
- sqrt((double) n) chiamata di sqrt su n int convertito a double (valore di n immutato!)

Precedenza e associatività

! ++ -- + - (tipo)	da destra a sinistra
* / %	da sinistra a destra
+ -	da sinistra a destra
< <= > >=	da sinistra a destra
== !=	da sinistra a destra
&&	da sinistra a destra
||	da sinistra a destra
= += -= *= /= %=	da destra a sinistra

C: Passaggio di array a funzioni (1)

- E' possibile passare un intero array ad una funzione
- Esempio

```
int max ( int v[4] ) ;
…
int w[4] ;
…
a = max( w ) ;
```

- In questo caso *gli elementi di w possono essere modificati da max. Passaggio di un array avviene per riferimento: viene passato l'indirizzo del primo elemento dell'array*

C: Passaggio di array a funzioni (2)

- E' opzionale specificare la dimensione dell'array nella dichiarazione/definizione di una funzione che ha un un array monodimensionale nella lista-argomenti,
- Esempio

```
int max ( int v[ ] ) ;
```

- In caso di array bidimensionale si deve specificare almeno la seconda dimensione
- Esempio

```
min ( int v[ ][7] ) ;
```


C: Funzione ricerca elemento in un array

```
int searchd( double [ ] , unsigned , double ) ;

int searchd( double a[ ] , unsigned dim , double w )
  {
  unsigned int h ;
  int found = -1 ;
  for ( h = 0 ; ( h < dim ) && ( found == -1 ) ; h++ )
      {
      if ( a[ h ] == w ) found = h ;
      }
  return found ;
  }

/* possibile chiamata  funzione searchd con b array di
   almeno 4 double  e posizione di tipo int*/
posizione = searchd( b , 4 , 11.322 )
```


C: Funzione ricerca valore minimo

- `double min( double [ ] , unsigned ) ;`
-
```
double min( double a[ ] , unsigned dim ) {
/* si assume dim >= 1 */
unsigned int h ;
double mini = a[ 0 ] ;
for ( h = 1 ; h < dim ; h++ ) {
    if ( a[ h ] < mini ) mini = a[ h ] ;
    }
return mini ;
}
```

- `z = min( b , 4 ) /* possibile chiamata  funzione min con b array di almeno 4 double e z double */`

C: Funzione ricerca indice valore minimo

- `unsigned int min( double [ ] , unsigned int ) ;`

- ```
  unsigned int min( double a[ ] , unsigned int dim ) {
  /* si assume dim >= 1 */
  unsigned int h , pos = 0 ;
  double mini = a[ 0 ] ;
  for ( h = 1 ; h < dim ; h++ ) {
        if ( a[ h ] < mini ) {
      mini = a[ h ] ;
      pos = h ;
       }
      }
  return pos ; /* pos = 1a posizione valore minimo */
  }
  ```


C: Funzione ricerca valore massimo

- ```
  int max(int [M1][M2], unsigned, unsigned ) ;
  /* M1, M2: massimo numero di righe, colonne */
  ```

- ```
  int max(int a[M1][M2], unsigned dim1, unsigned dim2)
  {  /* si assume dim1 >= 1 , dim2 >= 1 */
  unsigned int h , k ;
  int max = a[ 0 ][ 0 ] ;
  for ( h = 0 ; h < dim1 ; h++ ) {
   for ( k = 0 ; k < dim2 ; k++ ) {
          if ( a[ h ][ k ] > max )
         max = a[ h ][ k ] ;
           }
   }
  return max ;
  }
  ```
  ```