Laboratorio di informatica

Ingegneria meccanica

Lezione 7 – 19 novembre 2007

Concetto di modulo di un programma (1)

- Lo sviluppo di programmi complessi avviene tipicamente per composizione di moduli, ognuno dei quali esegue uno compito semplice
- Con l'uso della modularizzazione (scomposizione di un programma in moduli) si ottengono normalmente programmi facilmente
  - testabili/mantenibili (sono possibili operazioni di test/modifica limitate a singoli moduli)
  - leggibili/documentabili (il programma si presenta come una composizione di compiti semplici)

(… e si semplifica la ripartizione di un progetto tra più progettisti)

Concetto di modulo di un programma (2)

- La modularizzazione è particolarmente apprezzata quando almeno un modulo è usato più volte durante l'esecuzione del programma (tipicamente, in questo caso si ha anche una compattazione del codice sorgente)
- E' utile tentare di anticipare la modularizzazione alla fase di sviluppo dell'algoritmo per la soluzione di un problema

Funzioni in C (1)

- Un programma C si compone di funzioni
- E' disponibile una collezione *predefinita* di funzioni che possono essere usate *direttamente* in ogni programma (libreria standard)
- Per introdurre una nuova funzione occorre specificare:
  - la sequenza di operazioni da eseguire
  - gli argomenti su cui operare
  - il risultato da restituire
- Un programma C viene modularizzato con l'uso di funzioni

Funzioni in C (2)

- Le istruzioni specificate in una funzione sono eseguite quando la funzione viene attivata
- La funzione main è automaticamente *attivata* quando il programma viene messo in esecuzione
- Le altre funzioni sono *attivate* durante l'esecuzione del programma, tramite una chiamata di funzione (call)
- Una funzione può *chiamare* una o più funzioni
- Una funzione può *chiamare* se stessa (chiamata ricorsiva)
- La funzione chiamante perde *temporaneamente* il controllo del flusso di esecuzione del programma (ceduto alla funzione chiamata) e lo recupera al termine dell'esecuzione della funzione chiamata

C: Introduzione di una nuova funzione

- Nel programma, una funzione può essere chiamata solo dopo averla dichiarata
- La dichiarazione di una funzione avviene per mezzo di un prototipo di funzione nel quale vengono specificati nome, argomenti e risultato della funzione
- La sequenza di istruzioni eseguite da una funzione è invece specificata nella sua definizione
- La chiamata di una funzione avviene specificando nome e argomenti, conformemente a quanto dichiarato nel prototipo della funzione

Esempio di funzione C (1)

- Cubo di un intero: argomento int, risultato int
- Dichiarazione: nome, argomenti, tipo del risultato

  int cubo (int) ;
- Chiamata: specificare nome, argomento

  a = cubo (b) ; (assegnerà ad a, qui supposto di tipo int, il valore prodotto dalla funzione cubo applicata al valore che b ha al momento della chiamata)
- Definizione: specificare nome, argomenti, risultato

```
int cubo ( int n )
{
  return n*n*n ; (restituzione valore prodotto)
}
```


Esempio di funzione C (2)

```
…
int cubo( int ) ;   /* prototipo funzione cubo */

int main( )
{
   int num ;
   …          /* attivazione funzione cubo */
   printf( "cubo(%d) = %d", num , cubo( num ) ) ;
   …
   return 0;
}
      /* definizione funzione cubo */
int cubo( int n )
{
   return n*n*n ;
}
```

C: Prototipo di funzione

- Formato:
  tipo-restituito nome (lista-argomenti) ;
- tipo-restituito è il tipo del valore restituito dalla funzione come risultato; se la funzione non restituisce alcun valore, usare void (si può restituire un solo valore del tipo specificato)
- nome è utilizzato nella chiamata e nella definizione: stesse regole date per il nome di una variabile
- lista-argomenti specifica un tipo e, opzionalmente, un nome per ogni argomento della funzione; una virgola separa i vari argomenti; può essere vuota (nessun argomento)

C: Definizione di funzione

- Formato (deve essere *coerente* con il prototipo)
  tipo-restituito nome (lista-argomenti)
  { corpo-funzione }
- tipo-restituito e nome: vedi prototipo
- lista-argomenti specifica un tipo ed un nome per ogni argomento della funzione; una virgola separa i vari argomenti; può essere vuota; *numero e tipo di argomenti e loro ordine come nel prototipo*
- corpo-funzione specifica le azioni da eseguire: contiene dichiarazioni di variabili, istruzioni, ecc. Per restituire un valore si deve usare (*almeno una volta*)
  return espressione ;

C: Chiamata di funzione

- In generale, la chiamata di una funzione si presenta inclusa in un'espressione, che è, a sua volta, inclusa in un'istruzione (vedi esempi avanti)
- Formato della chiamata
  nome (lista-espressioni) (deve essere *coerente* con il prototipo della funzione stessa)
  - nome: vedi prototipo
  - lista-espressioni contiene un'espressione per ogni argomento; espressioni separate da virgola; *numero e tipo di espressioni e loro ordine come nel prototipo*

C: Chiamata di funzione (2)

- Esempio di chiamata di funzione

```
z = y - v*cubo( t*w + x ) ;
```

- E' un'istruzione di assegnazione dove nell'espressione da valutare per l'assegnazione compare la chiamata della funzione cubo
- E' sensato ipotizzare che cubo sia stata dichiarata e definita specificando 1 argomento, di tipo compatibile con la corrispondente espressione usata nella chiamata (t*w + x), e la restituzione di un valore, di tipo compatibile con l'espressione in cui la chiamata è inserita

C: Chiamata di funzione (3)

- Esempio di chiamata di funzione

```
if ( z == fact( u , v )) printf("OK!") ;
```

- E' un'istruzione if dove nell'espressione da valutare per decidere se eseguire o no l'istruzione printf("OK!") ; compare la chiamata della funzione fact
- E' sensato ipotizzare che fact sia stata dichiarata e definita specificando
  - la restituzione di un valore, di tipo compatibile con l'espressione in cui la chiamata è inserita
  - l'uso di 2 argomenti, di tipi compatibili con u e v, rispettivamente

C: Chiamata di funzione (4)

- Esempio di chiamata di funzione

```
w = fff( ) ;
```

- E' un'istruzione di assegnazione dove l'espressione da valutare per l'assegnazione è costituita dalla chiamata della funzione fff
- E' sensato ipotizzare che fff sia stata dichiarata e definita specificando l'uso di nessun argomento e la restituzione di un valore, di tipo compatibile con w

C: Chiamata di funzione (5)

Esempio di chiamate di funzioni

```
printf( "%f %f %f", x , f( x ) , g( x )) ;
```

E' un'istruzione di visualizzazione dove le espressioni da valutare per fornire due dei tre valori da visualizzare sono costituite dalle chiamate delle funzioni f e g

E' sensato ipotizzare che f e g siano state dichiarate e definite specificando 1 argomento, di tipo compatibile con x, e la restituzione di un valore, di tipo float

C: Chiamata di funzione (6)

- Esempi di chiamate di funzioni

```
funct1( alpha ) ;
test((3*g - beta*beta) , 2*h ) ;
```

Sono istruzioni costituite dalle chiamate delle funzioni funct1 e test, rispettivamente

E' sensato ipotizzare che funct1 e test siano state dichiarate e definite specificando
- la restituzione di nessun valore
- l'uso di 1 e 2 argomenti, rispettivamente, di tipi compatibili con le corrispondenti espressioni usate nelle chiamate (alpha e (3*g - beta*beta) e 2*h, rispettivamente)

C: Chiamata di funzione (7)

Esempio di chiamata di funzione

```
ggg( ) ;
```

- E' un'istruzione costituita dalla chiamata della funzione ggg
- E' sensato ipotizzare che ggg sia stata dichiarata e definita specificando la restituzione di nessun valore e l'uso di nessun argomento

C: prototipo, definizione, chiamata (1)

```
• double power( double , unsigned int ) ;
• double power( double base, unsigned int expo )
  {
    int h ;
    double result = 1 ;
    for( h = 1 ; h <= expo ; h++ )
     result = result * base ;
    return result ;
  }
• y = power( x , n ) ;
```


C: prototipo, definizione, chiamata (2)

```
• void power( double , unsigned int ) ;
• void power( double base, unsigned int expo )
  {
    int h ;
    double result = 1 ;
    for( h = 1 ; h <= expo ; h++ )
     result = result * base ;
    printf( "power(%lf,%u) = %lf" , base, expo, result ) ;
    return ;  /* return può essere omesso */
  }
• power( x , n ) ;
```


C: Programma con funzioni

Schema di riferimento *per questo corso*

- Dichiarare le funzioni tramite i prototipi prima del main
- Definire le funzioni corrispondenti ai prototipi dopo il main
- Inserire le chiamate delle funzioni dove necessario

C: Passaggio degli argomenti

- Gli argomenti presenti nel prototipo e nella definizione di una funzione vengono detti anche argomenti (o parametri) formali, distinguendoli così dagli argomenti effettivamente utilizzati durante la chiamata della funzione, detti argomenti (o parametri) attuali
- Nella chiamata power(x , n) gli argomenti attuali sono i valori delle variabili x ed n al momento della chiamata. Alla funzione vengono passati i *valori* di x ed n. La funzione non può modificare il valore di x ed n

C: Variabili locali

```
double power( double base, unsigned int expo )
{
  int h ;
  double result = 1 ;
  for( h = 1 ; h <= expo ; h++ )
   result = result * base ;
  return result ;
}
```

- Variabili h e result sono dette locali: non sono visibili al di fuori del blocco della funzione power
- Nel blocco della funzione power non sono visibili le variabili dichiarate all'interno della funzione main o di altre funzioni

Esempio: Funzione in un programma

```
int max( int , int ) ;            /* prototipo */
int main ( )
{
    int num1, num2 ;
    printf( "Inserire due interi\n" ) ;
    scanf( "%d%d" , &num1 , &num2 ) ;
    printf( "Max = %d\n ", max( num1 , num2 )) ;
    return 0 ;                    /* chiamata */
}
int max ( int n1 , int n2 )       /* definizione */
{
    if ( n1 > n2 ) return n1 ;
    else return n2 ;
}
```


Esecuzione di una funzione in C

L'incontro (durante l'esecuzione del corpo di una funzione) di una chiamata di funzione provoca un trasferimento del controllo del flusso di esecuzione delle istruzioni dal corpo della funzione chiamante al corpo della funzione chiamata, che viene eseguito assegnando agli argomenti della funzione chiamata i valori degli argomenti usati nella chiamata

Quando l'esecuzione della funzione chiamata termina, il controllo del flusso di esecuzione delle istruzioni ritorna al corpo della funzione chiamante

Si osservi che una funzione chiamata può a sua volta chiamare una funzione