Lezione 11 e 12

- Funzioni e prototipi
- Passaggio parametri
- Ricorsione
- Riuso del codice

Fabio Scotti (2004-2009)

Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)

Laboratorio di programmazione

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 11 e 12

Funzioni e prototipi

Obiettivi :

- Capire perchè è importante progettare il codice mediante funzioni e prototipi
- Capire cosa sia un prototipo di una funzione e cosa serve al compilatore

Obiettivi

- Costruire i programmi in modo modulare usando piccoli pezzi di codice chiamati funzioni.
- Introdurre nei programmi funzioni matematiche contenute nelle librerie standard del C.
- Creare nuove funzioni.
- Definire le modalità per passare informazioni alle funzioni.
- Leggere i risultati delle funzioni.

3

Funzione matematica

- L'uso più intuitivo delle funzioni è il loro impiego in ambito matematico.
- Consideriamo il seguente codice:

```
float x, y;
x = 3.14;
y = sin( x );
printf("il seno vale %f", y);
```

- La funzione sin elabora il valore passato della variabile x e restituisce il risultato, che viene salvato in y.

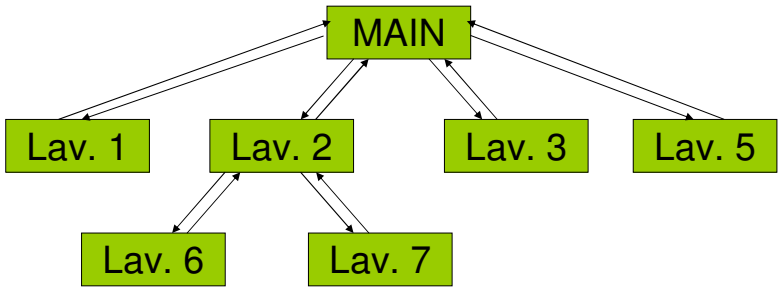
4

Definizione generica di funzione

- **In generale possiamo vedere le funzioni come porzioni di codice che ricevono dei dati da elaborare e ritornano dei risultati.**
- **Esiste una analogia: la relazione tra le funzioni e il programma (il main) è simile a quella esistente tra i lavoratori di una azienda e il loro capo.**
- **Il capo attribuisce ai lavoratori compiti e fornisce loro informazioni per svolgere tali compiti; i lavoratori ottengono (ritornano) i risultati.**

5

Analogia con una azienda



- **Il capo ignora il modo in cui i lavoratori hanno conseguito i risultati: information hiding.**
- **Esiste una gerarchia nelle chiamate.**

6

Terminologia delle funzioni

- **L'utilizzo delle funzioni in un programma è detto invocazione o chiamata.**

  Esempio: `y = sin(x) ;`

- **I dati passati alle funzioni sono i parametri.**

  Esempio: `pot = potenza(x, esp) ;`

- **La porzione di codice che descrive le istruzioni eseguite da una funzione è detta definizione.**

7

Funzioni matematiche

```
double acos()          // arco coseno
double asin()          // arco seno
double atan()          // arco tangente x
double atan2()         // arco tangente y/x
double cos()           // coseno
double cosh()          // coseno iperbolico
double exp()           // funzione esponenziale
double fabs()          // valore assoluto di un double
double log()           // logaritmo naturale
double log10()         // logaritmo base 10
double pow()           // potenza
double sin()           // seno
double sinh()          // seno iperbolico
double sqrt()          // radice quadrata
double tan()           // tangente
double tanh()          // tangente iperbolica
```

Esempio di utilizzo:

```
#include <math.h> // necessario
double x, y;
x = 0.1;
y = acos(x);
```

8

Prototipo, chiamata e definizione (1)

- **In un programma il corretto utilizzo di una funzione richiede che essa abbia le seguenti porzioni di codice: il prototipo, la chiamata e la definizione.**
- **Analizziamo significato e posizione nel programma mediante il seguente esempio:**
  - stampare a monitor le potenze seconde dei seguenti dieci interi: 1, …, 10;
  - ai fini della soluzione utilizzare un ciclo for con una printf e costruire la funzione **square()**.

9

Prototipo, chiamata e definizione (2)

```
#include <stdio.h>
int square( int y );   // prototipo

int main()
{
    …
}

int square( int y )                    // definizione
{
    return ( y * y );
} // fine della funzione square
```

10

Prototipo, chiamata e definizione (3)

```
#include <stdio.h>
int square( int y );   // prototipo
int main()
{
    int x;
    for ( x = 1; x <= 10; x++ )
        printf( "%d ", square( x ) ); // chiamata
    printf( "\n" );
    return 0;
}
int square( int y )  // definizione
{
    return ( y * y );
} // fine della funzione square
```

11

Altre nozioni sul prototipo (1)

Il prototipo serve al compilatore per individuare:

- il **numero di parametri** della funzione;
- il **tipo di parametri** della funzione;
- l'**ordine dei parametri** della funzione.

Esempio:

```
// int square ( int x );
// int massimo( int a, int b );

// int square ( int );      // anche questo e' OK
// int massimo( int , int );
```

12

Altre nozioni sul prototipo (2)

- **E' regola di buona programmazione riportare tutti i prototipi delle funzioni utilizzati nel programma.**
- **Il compilatore può trovare i prototipi utilizzati inclusi in un file header.**

```
#include <math.h>
int main()
{
    …
    y = sin(x);
    z = log(y);
}
```

Nell'esempio riportato i prototipi sono nel file header **math.h**

- **Le definizioni possono essere presenti in file header oppure in librerie di funzioni precompilate.**

13

Altre nozioni sulla definizione

- **La definizione di una funzione indica al compilatore quali istruzioni devono essere eseguite.**
- **La definizione inizia con una intestazione del tutto simile al prototipo ma senza il ';'.**

Esempio:

```
…
int square( int y );        prototipo
…
int square( int y )         intestazione
{                                          definizione
    return ( y * y );
} /* fine della funzione  square */
```

14

Struttura della definizione

La definizione di una funzione deve essere composta nel seguente modo:

```
tipo_Del_Valore_Ritornato nome_Della_Funzione( lista_Dei_Parametri )
        {
        dichiarazioni

        istruzioni

        } /* fine della funzione */
```

intestazione

Parametri formali e variabili locali

La lista di parametri nella definizione della funzione esprime quali variabili possono essere usate nella funzione.

Esempio:

```
int massimo (int a, int b )
{
  int max;
  if (a > b)
        max = a;
  else
        max = b;
  return max
}
```

parametri formali

variabile locale

Regole di visibilità

- **Dentro la funzione si "vedono" solo le variabili locali, i parametri formali e le variabili globali.**
- **Al di fuori dalla funzione NON si vede alcuna variabile della funzione (né i parametri formali né le variabili locali).**
- **Le regole di buona programmazione sconsigliano di usare variabili globali: tutte le funzioni possono accedere in scrittura a tutte le variabili globali, pertanto può risultare difficile trovare gli errori.**

17

Funzioni e procedure

- **Le funzioni che non ritornano valori sono chiamate procedure.**
- **Nel C si usano solo funzioni. Una procedura viene quindi dichiarata usando "`void`".**

Esempio:

```
void stampa( int y );   // prototipo di procedura
int main()
{ …
    stampa( x ) ; // chiamata
…}
void stampa( int y )  // definizione di procedura
{
    printf("il numero passato e' %d", y);
} // fine della procedura stampa
```

18

Punti di attenzione

- **Lunghezza di una funzione:**
  - lunghezza massima: non più di una pagina!
  - lunghezza migliore : mezza pagina!
- **Se una funzione diventa troppo lunga spezzarla in più funzioni.**
- **Le funzioni aumentano la riusabilità del codice.**
- **Se una funzione richiede un elevato numero di parametri significa che fa troppe cose: spezzarla!**
- **L' intestazione di una funzione dovrebbe essere lunga non più di una riga!**

19

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 11 e 12

Passaggio parametri

Obiettivi :

- Capire il funzionamento del passaggio parametri, per valore e per indirizzo, alle funzioni
- Capire come si passano alle funzioni i vettori, le matrici e tipi di dato strutturato
- Essere in grado di passare correttamente i parametri alle funzioni

PASSAGGIO PER VALORE

Passaggio dei parametri

- **In C, come in molti altri linguaggi, esistono due modi per passare i parametri alle funzioni:**
  - per valore;
  - per indirizzo.
- **Il passaggio per valore COPIA il valore dei parametri con cui è chiamata la funzione nei parametri formali.**

  Esempio:

```
float x, y;
x = 3.14;
y = sin(x);
```

Passaggio parametri per valore

- **Il passaggio per valore si usa quando la funzione chiamata non deve modificare i parametri che le vengono passati.**

  Esempio: `y = sin(x); // non deve modificare x!`

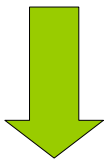
- **Il passaggio per valore evita effetti collaterali (SIDE EFFECT) se sono presenti errori nelle funzioni perchè nulla accade ai parametri passati.**

  Esempio:
  ```
  y = elaboro(x,z,k,…);
          // nemmeno volendo posso modificare
          // x, z, k
  ```

23

Falso SWAP (1)

- **Il passaggio per valore (copia).**
- **Non è possibile modificare il valore dei parametri con cui viene chiamata la funzione.**

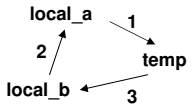


Scrivere un programma che scambi il valore di due variabili (**`a`** e **`b`**) usando una funzione con passaggio parametri per valore (**`swap(a, b)`**)

24

Falso SWAP (2)

```
#include <stdio.h>
#include <stdlib.h>

void swap1( int local_a , int local_b );

int main()
{
        int a;    int b;
        a = 3;    b = 5;
        swap1( a, b);
        printf("a vale %d  e b vale %d \n" , a , b);
        exit(0);
}

void swap1( int local_a , int local_b )
{   int temp ;
    temp = local_a ;
    local_a = local_b ;
    local_b = temp ;
}
```



25

Falso SWAP (3)

```
#include <stdio.h>
#include <stdlib.h>

void swap1( int a , int b );

int main()
{
        int a;    int b;
        a = 3;    b = 5;
        swap1( a, b);
        printf("a vale %d  e b vale %d \n" , a , b);
        exit(0);
}

void swap1( int a , int b )  // non funziona ugualm.
{   int temp ;
    temp = a ;
    a = b ;
    b = temp ;
}
```

Le variabili locali:

- **sono indipendenti da quelle nel main;**
- **cessano di esistere al termine dell'ultima istruzione della funzione**

26

Funzione fattoriale (1)

Esempio di progettazione del codice usando la funzione fattoriale di un numero intero:

- **il programma sarà basato su una funzione `fattoriale()`**
- **innanzitutto si progetta la funzione definendo gli ingressi e il loro tipo, ossia si scrive la definizione della funzione:**

```
int fattoriale( int n )
```

- **il fattoriale si calcola sui numeri interi ed il risultato è ancora un numero intero.**

27

Funzione fattoriale (2)

```
…include…
int calcola_fattoriale(int n); // prototipo

void main()
{
    int n, fatt;
    printf("intero---->");   scanf("%d", &n);

    fatt=calcola_fattoriale(n); // chiamata per valore
    printf("\n Il fatt. di %d e' l'intero %d\n",n,fatt);
}

int calcola_fattoriale(int n)  // intestaz. e definiz.
{
    int i, fatt=1;
    for (i=1; i<=n; i++)
    {
        fatt = fatt * i;
    }
    return(fatt); // ritorna il valore calcolato
}
```

28

Funzione fattoriale (3)

- **La funzione `fattoriale(n)` deve ricevere il numero intero `n` ma non deve modificarlo, quindi il passaggio parametri per valore è adeguato.**
- **Il numero ritornato da `fattoriale()` è stato stabilito sia un intero, ma lo spazio di rappresentazione (il numero maggiore rappresentabile) potrebbe essere insufficiente.**
- **Per ovviare, definire il valore ritornato come `long` o meglio ancora come `double`:**

```
double fattoriale( int n)
```

29

Progettare funzioni

- **La fase di progettazione della funzione è fondamentale.**
- **Esempi di intestazione:**
  - calcolare la potenza di due reali

    ```
    float potenza( float x, float esponente)
    ```
  - stampare 3 interi

    ```
    void stampainteri( int a, int b, int c)
    ```
  - stampare il logo aziendale

    ```
    void stampalogo ( void )
    ```
  - calcolare pigreco con 8 cifre dopo la virgola

    ```
    float calcolaPI ( int cifre )
    ```

30

Main: procedura vs. funzione

```
void main()
{
   … istruzioni
}
```

il `main()` inteso come **procedura** che non ritorna nulla al sistema operativo che ha messo in esecuzione il programma

```
int main()
{
   … istruzioni
   return 0;
}
```

il `main()` inteso come **funzione** che ritorna 0 al sistema operativo (**STANDARD ANSI C**)

31

Coercizione degli argomenti in C

- **Il C esegue una coercizione forzata degli argomenti delle funzioni al tipo più appropriato.**
- **Esempio: la funzione `sqrt` della libreria `math.h` ha un prototipo del tipo `float sqrt (float);` Cosa succede se si passa ad un intero?**

```
printf("%.3f \n"  , sqrt(4) );
```

```
2.000
```

- **La coercizione degli argomenti converte 4 (intero) nel 4.000 (float). Si noti che il risultato è ancora un float (2.000).**

32

Regole di promozione in C

- **Il C esegue una conversione del tipo delle variabili quando si hanno tipi di dato misti nella stessa espressione (espressione mista).**

  Esempio: `… x_long + 3 + 2.00 + sqrt(3);`

- **In un'espressione mista ogni valore sarà automaticamente promosso al valore più "forte" dell'espressione (in realtà verrà creata una copia temporanea di ognuno dei valori, lasciando gli originali invariati).**
- **In particolare:**

  `double` è più forte di `float` che è più forte di `long` che è più forte di `int` che è più forte di `short`.

33

PASSAGGIO PER INDIRIZZO

34

Passaggio parametri per indirizzo

- **Il passaggio per indirizzo NON COPIA il valore dei parametri ma copia l'indirizzo della variabile passata.**

  Esempio:

```
int x;

x = 2;
raddoppia(&x);   // &x e' indirizzo di x

printf("%d", x); // stampa 4
```

- **La funzione avendo ha il vero indirizzo della variabile passata (e non una copia) quindi può accedere ad essa in lettura e SCRITTURA.**
- **Attenzione ai side effect degli errori di programmazione.**

35

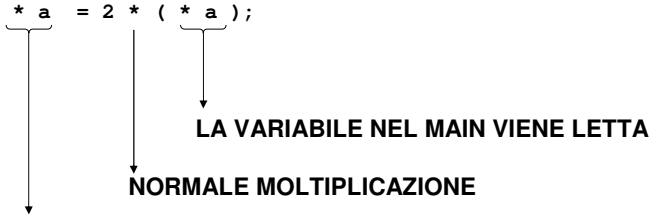
Progettazione della funzione (1)

```
…#include …

void raddoppia ( int * a ); // prototipo

int main()
{
       int x;
       x = 2;
       raddoppia(&x);    // &x e' indirizzo di x
       printf("%d", x); // stampa 4
}

void raddoppia ( int * a )
{
       int temp ;
       temp = * a ;      // leggo la variabile nel main
       * a  = 2 * temp ;// scrivo la variabile nel main
}
```

36

Progettazione della funzione (2)

```
// oppure piu' in modo compatto posso scrivere

void raddoppia ( int * a )
{
        * a  = 2 * ( * a );
}
```



37

Vero SWAP (1)

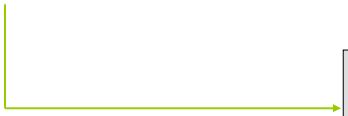
```
#include <stdio.h>
#include <stdlib.h>

void swap_OK( int * i_a , int * i_b );

int main()
{
    int a;
    int b;

    a = 3; b = 5;

    printf("a vale %d  e b vale %d \n" , a , b);
    swap_OK( &a, &b);
    printf("a vale %d  e b vale %d \n" , a , b);

    exit(0);
}

void swap_OK( int * i_a , int * i_b )
{
    int temp ;
    temp = *(i_a) ;
    *(i_a) = *(i_b) ;
    *(i_b) = temp ;
}
```

38

Vero SWAP (2)

```
void swap_OK( int * i_a , int * i_b )
{
    int temp ;
    temp = *(i_a) ;
    *(i_a) = *(i_b) ;
    *(i_b) = temp ;
}
```

```
printf("a vale %d  e b vale %d \n" , a , b);
swap_OK( &a, &b);
printf("a vale %d  e b vale %d \n" , a , b);
```



```
a vale 3  e b vale 5
a vale 5  e b vale 3
```

39

Passare gli array alle funzioni (1)

- **In C gli array vengono passati SEMPRE PER INDIRIZZO.**
- **Le funzioni non dispongono di una copia degli array ma lavorano sull'originale, quindi occorre prestare attenzione ai side effect.**

Esempio di prototipo:

```
void stampaArray( int a[], int lung);
```

- mediante le parentesi [] il compilatore capisce che si passa un array;
- si passa sempre la lunghezza degli elementi dell'array.

40

Passare gli array alle funzioni (2)

Esempio:

```
void stampaArray( int a[], int lung);


int main()
{
   int a[5];
   a[0]=1; a[1]=2; a[2]=3; a[3]=4; a[4]=5;
   …
   stampaArray ( a , 5);
   …          // Attenzione a e' l'indirizzo!
```

41

Passare le stringhe alle funzioni (1)

- **Le stringhe sono array di caratteri, quindi seguono le regole degli array.**
- **Le stringhe vengono passate per indirizzo: attenzione ai side effect.**
- **Le stringhe devono sempre avere in coda il carattere terminatore '\0', quindi è possibile omettere la lunghezza della stringa.**
- **Esiste le funzione `strlen()` che ritorna la lunghezza della stringa passata.**

42

Passare le stringhe alle funzioni (2)

Esempio:

```
char stringa[32];
…
printf ( "%s" , stringa);
```

- **Si consideri il seguente prototipo della funzione:**

```
void inizializza( char stringa[] );
```

- **Il prototipo della funzione nel main verrà chiamato nel seguente modo:**

```
inizializza( stringa );
```

- **La funzione può modificare la stringa `stringa` dichiarata nel main.**

43

Passare le matrici alle funzioni (1)

- **Le matrici sono array N-dimensionali con N=2 di elementi.**
- **Le matrici vengono passate per indirizzo: attenzione ai side effect.**
- **In generale per gli array N-dimensionali è necessario passare le N-1 dimensioni alla funzione.**

Esempi di prototipi:

```
void stampaMatrice( int a[][5] );
void stampaArray3D( int a[3][5][4] );
void stampaArray4D( int a[][5][6][5] );
```

44

Passare le matrici alle funzioni (2)

Esempio:

```
#define LUN 5
void stampaMatrice( int a[][LUN] );
…
int main()
{
      int a[LUN][LUN];
      a[0][0]=1; …

      riempi ( a );
      stampaMatrice ( a );
      … }
```


Passare gli `struct` alle funzioni (1)

- **Gli struct si possono passare per indirizzo, ma anche per valore.**

  Come avevamo scritto per gli interi:

```
void stampaintero( int i);      // per copia (valore)
void modificaintero( int * i ); // per indirizzo
```

  Similmente:

```
void stampaDatoStruct( tipoStruct nome);
                                // per copia (valore)

void inserisci_miss ( tipoStruct * nome);
                                // per indirizzo
```

Passare gli `struct` alle funzioni (2)

- **Nel passaggio per indirizzo, all'interno della funzione si accede ai campi della variabile struct usando la NOTAZIONE '->'**

  Esempio:

```
void modificaDatoStruct( tipoStruct * p)
{
numero = p->numero;
printf("nome = %s", p->nome );
…
}
```


Passare gli `struct` alle funzioni (2)

```
typedef struct
       {
              char nome[256] ;
              char cognome[256] ;
              int voto[3];
              int altezza;
              int peso;
              int cucina;
       } MissItalia;

void inserisci_miss ( MissItalia * ragazza ) ;

void stampa_miss    ( MissItalia ragazza  );

int main()
{
       MissItalia ragazza1;
```

Passare gli `struct` alle funzioni (3)

```
        inserisci_miss ( &(ragazza1) );
        stampa_miss    ( ragazza1  );

…
} // fine main


void inserisci_miss ( MissItalia * ragazza ) // per indirizzo
{
        …
        ragazza->altezza   = 180;
        ragazza->voto[0] = 80;
        …
}

void stampa_miss ( MissItalia ragazza )  // per copia
{
        …
        printf("%d", ragazza.altezza);
        printf("%d", ragazza.voto[0] );
        …
}
```

49

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 11 e 12

Ricorsione

Obiettivi :

- Comprendere l'uso della ricorsione come strumento di programmazione
- Identificare i punti di forza e di debolezza della ricorsione
- Essere in grado di impiegare la ricorsione correttamente nei propri programmi

L'idea della ricorsione

- **Nei programmi visti le funzioni eseguono le loro istruzioni ed al massimo chiamano altre funzioni.**
- **Una funzione si dice ricorsiva quando richiama sé stessa direttamente oppure attraverso un'altra funzione.**
- **Alcune classi di problemi anche molto complessi posso essere risolti da semplici programmi ricorsivi.**

51

Problemi adatti alla ricorsione

- **I problemi adatti ad essere risolti con la ricorsione possono essere espressi nel modo seguente:**
  - Un caso o più casi banali che hanno una soluzione fissata;
  - Uno o più casi complessi che possono essere risolti tramite il risultato di un problema più semplice.
- **Esempio: il fattoriale**

  n! = n * (n - 1)* … * 3 * 2 * 1
  con il caso 0! = 1.
- **Soluzione ricorsiva:**

  0! = 1 **(caso banale)**
  n! = n * (n-1)! **(se n > 0).**

52

Fattoriale non ricorsivo

```
#include <stdio.h>

int calcola_fattoriale(int n);

int main()
{
    int n, fatt;
    printf("intero---->");  scanf("%d", &n);

    fatt=calcola_fattoriale(n);

    printf("Il fattoriale di %d e' %d\n",n,fatt);
}


int calcola_fattoriale(int n)

{
    int i, fatt=1;
    for (i=1; i<=n; i++)
    {
        fatt = fatt * i;
    }
    return(fatt); // ritorna il valore calcolato
}
```

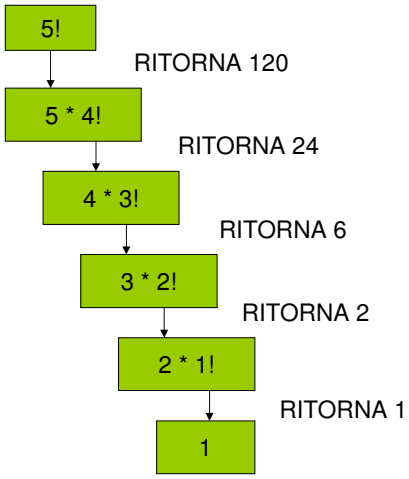

Fattoriale ricorsivo

- **Non è necessario cambiare il main ma solo la funzione.**

```
int calcola_fattoriale(int n)

{
    int risultato;

    if ( n == 1 )            // base della ricorsione

          risultato=1;
    else                     // passo ricorsivo

          risultato = n * calcola_fattoriale(n-1);


    return(risultato);

}
```

Struttura delle chiamate (II)

Seguiamo la sequenza di chiamate nel caso
`calcola_fattoriale(5)`



Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 11 e 12

Modalità di riuso del codice in una funzione

Obiettivi :

- Capire le modalità per incapsulare codice già scritto in altre funzione
- Essere capaci di riusare codice già scritto nelle funzioni
- Capire perchè è importante progettare il codice mediante funzioni e prototipi

Incapsulare codice nelle funzioni

- **Porzioni di codice usati in molti punti possono essere racchiusi in funzioni in modo che sia più agevole usarle e correggerle.**
- **Il problema è trovare un passaggio parametri corretto, in modo tale che il codice continui a funzionare.**
- **Evitare l'uso di variabili globali rende l'incapsulamento più agevole e meno sottoposto a rischio di errore.**

Idea di fondo

- **Si immagini di spostare il codice da incapsulare in una funzione.**
- **Tutte le variabili che servono a questa porzione di codice e che vengono solo lette, verranno passate per copia alla funzione.**
- **Alla fine della porzione di codice che vogliamo incapsulare, tutte le variabili che sono state modificate (e che servono fuori dalla funzione) verranno passate alla funzione per indirizzo e impiegate con '*'.**

Esempio di conteggio (1)

```
#include <stdio.h>
#include <string.h>

int main()
{
        char stringa[128]  ;
        int i, len, numeroa;


        scanf( "%s", stringa );
        len = strlen(stringa);

        numeroa = 0;
        for (i=0; i<len; i++)
        {
         if (stringa[i]=='a') numeroa = numeroa+1;
        }

        printf("il numero di a e'= %d", numeroa);

        getchar();     exit(0);

}  // main
```

59

Esempio di conteggio (2)

```
#include <stdio.h>
#include <string.h>

void contaa( char stringa[], int len, int * numeroa);

int main()
{
        char stringa[128]  ;
        int  len, numeroa;


        scanf( "%s", stringa );
        len = strlen(stringa);

        contaa( stringa, len, &numeroa);

        printf("il numero di a e'= %d", numeroa);

        getchar();     getchar();
        exit(0);

}  // fine main
```

60

Esempio di conteggio (3)

```
…
     contaa( stringa, len, &numeroa);
…
} // fine main

void contaa( char stringa[], int len, int * numeroa)
{
    int i;

    * numeroa = 0;

    for (i=0; i<len; i++)
    {
         if (stringa[i]=='a') *numeroa = *numeroa + 1;
    }
}
```

Nota Bene:

- **`len` è passata per copia;**
- **`numeroa` è passata per indirizzo.**

61