Le funzioni

Funzioni

- Con il termine *funzione* si intende, in generale, un operatore che, applicato a un insieme di operandi, consente di calcolare un risultato, come avviene anche per una funzione matematica f(x).
- f(x) restituisce un valore, in generale, diverso per ogni diverso valore di x. I valori che inseriamo tra parentesi, che in matematica chiamiamo *variabili indipendenti*, in C si chiamano *parametri della funzione*

Funzioni

- In C, per *definire una funzione,* occorre prima *dichiarare* il cosiddetto *prototipo* della funzione nella sezione delle dichiarazioni globali (fuori dal main(), insieme alle variabili che possono essere viste da tutti i blocchi di codice che costituiscono il programma):

<tipo risultato> <nome funzione> (<elenco parametri>);

dove:
<elenco parametri> ::=
<tipo parametro> [<nome parametro>]
{,<tipo parametro> [<nome parametro>]}

Funzioni

- Dichiarare una funzione permette al compilatore di fare un controllo sul tipo dei parametri che le verranno poi passati quando verrà usata

Esempi:
```
int somma (int m, int n);
int somma (int, int);
int fun(void);
```

- void è un tipo di dato speciale che rappresenta *assenza di valori:* quindi fun è una funzione che restituisce un valore intero ma non in dipendenza di un *parametro,* ma di ciò che avviene al suo interno durante l'esecuzione (es. lettura di un dato).

NB *Non è obbligatorio dare un nome ai parametri nella dichiarazione di una funzione: basta il tipo. Le due dichiarazioni della funzione somma sono entrambe valide*

Funzioni

- Occorre poi *definire* la funzione

<tipo risultato> <nome funzione> ([<elenco par.>])
{
<corpo della funzione>
}

- All'interno del corpo della funzione si utilizza un'apposita istruzione per terminare l'esecuzione della funzione e restituirne il risultato:
return <espressione risultato>

Funzioni

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

int doppio(int);  /* dichiarazione funzione */

int doppio(int x){  /* definizione funzione */
return ( 2 * x );
}

 int main(){    /* programma principale che */
 int d, p=5;    /* usa la funzione */
 d = doppio(p);
 printf("%d", d);
 return 0;   /* anche main() è una funzione */
}
```

Esempio 2

```
#include <stdio.h>

int leggi_numero (void);
int max ( int m, int n );

int main () {
  int a, b, massimo;
  a = leggi_numero();
  b = leggi_numero();
  massimo = max(a,b);
  printf ("Il massimo è %d\n",massimo);
  return 0;
}
```


Esempio 2 (continuazione)

```
/* Funzione senza parametri */
int leggi_numero(void) {
  int n;
  printf ("Inserire un numero:");
  scanf ("%d", &n);
  return n;
}

int max (int m, int n) {
  if (m>n)  return m;
  else  return n;
}
```


Funzioni con parametri

- Possono operare su valori diversi per ogni *chiamata di funzione*, a seconda del valore che assumono in quel momento le espressioni che sono inserite nella lista dei parametri
- Si dice che i parametri vengono passati *per valore*

Passaggio parametri *per valore (by value)*

- **Parametri formali** utilizzati nella definizione della funzione: indicano *come* operare sui valori che vengono passati nella chiamata ma 'non esistono' al di fuori della funzione.
- La funzione che utilizza al suo interno un'altra funzione si dice, rispetto ad essa, *funzione chiamante;* l'utilizzo di una funzione si dice *chiamata di funzione*.
- La funzione opera *sui valori senza alterare la variabile* eventualmente utilizzata nella chiamata.
- Ogni chiamata deve passare tanti valori, attraverso una serie di espressioni dello stesso tipo dei parametri corrispondenti, quanti sono i parametri formali.

Passaggio parametri *per valore (by value)*

- **Parametri effettivi** passati per valore dalla *funzione chiamante*
  - I valori dei dati effettivi sono copiati nei parametri formali utilizzati dalla funzione chiamata
  - Nessun effetto provocato da modifiche nel parametro formale all'interno della funzione si ripercuote sul parametro reale del programma chiamante

Passaggio parametri *per valore (by value)*

```
#include<stdio.h>
int doppio(int);

int main() {
int g = 5, h;
h = doppio(g);
printf("%d %d", g, h);
return 0;
}

int doppio(int x) /* x è un parametro formale */
{return ( 2 * x );}
```

Quale sarà il valore stampato per le variabili g e h ?

Esempio

- Si realizzi un programma che legga da tastiera il costo di listino di un prodotto e la percentuale di sconto e visualizzi il prezzo da pagare
- Si può realizzare una funzione che riceva come parametri i due valori e calcoli il prezzo finale

Esempio

```
#include<stdio.h>

/* Definizione funzione con parametri*/
long prezzo_scont (long valore, int percent) {
/*Definizione interna alla funzione*/
long int val_scont;

val_scont=valore-(valore*percent/100);

return (val_scont);
}
```

Valore e percent sono i *parametri formali* della funzione
NB 'long' e 'long int' sono sinonimi.

Esempio

```
void main() {
long int costo;
int percentuale;

printf("Introduci costo e percentuale (interi):\n");
scanf("%ld%d",&costo,&percentuale);

printf("Prezzo di listino: %ld, sconto: %d%%",
                        costo, percentuale);
printf("prezzo finale: %ld \n",
                prezzo_scont(costo, percentuale));}
```

Costo e percentuale sono i *parametri effettivi*

Funzioni : il passaggio di array

- GIi array non sono passati per valore
- GIi array vengono passati *per indirizzo* (in realtà si tratta ugualmente di un passaggio per valore, in quanto in C il nome di un array rappresenta l'indirizzo del primo elemento)
- Quindi la funzione lavora realmente sulle variabili che si trovano a quell'indirizzo (quindi direttamente sull'array)
- Perciò, se la funzione modifica il contenuto dell'array, tale modifica si riflette sull'array originario
- Non occorre specificare la dimensione dell'array nell'elenco dei parametri formali (tanto il C non le controlla!)

Esempio

```
/* Restituisce il valore del minimo di un vettore di interi.
   vet: vettore di cui si cerca il minimo
   dim: numero di elementi del vettore */

#include<stdio.h>
int min_ele (int vet[], int dim);

int main(){
int i,numeri[10];
 for (i=0; i<10; i++){
          printf ("Inserire un numero:");
          scanf ("%d", &numeri[i]);
     }

printf("Il minimo e' : %d",min_ele(numeri,i);
return 0;}
```

Esempio (continuazione)

```
int min_ele (int vet[], int dim) {

  int i, min;
  min=vet[0];

  for (i=1;i<dim;i++)
     if (vet[i]<min) min=vet[i];

  return min;
}
```

Esercizi

- Scrivere una funzione
  int max(int vet[ ], int dim) che calcoli l'elemento del vettore vet di dimensione dim di valore più elevato.
- Utilizzando la funzione scrivere un programma che, dati 2 vettori:
  - stampi i valori degli elementi di ciascun vettore,
  - calcoli il valore più grande fra il massimo dell'uno e dell'altro
  - stampi il valore di tale massimo, indicando in quale dei due vettori è contenuto.

NB E' possibile inizializzare un vettore quando lo si dichiara, nel modo seguente:
int vet[5]={ 10, 27, 12, 1, 6 };

Soluzione

```
int max_ele (int vet[], int dim) {

  int i, max;
  max=vet[0];

  for (i=1;i<dim;i++)
     if (vet[i]>max) max=vet[i];

  return min;
}
```


Soluzione (main)

```
#include <stdio.h>
int max_ele (int vet[], int dim);

int main(){
 int i,numeri[4]={11,1,5,20},numbers[3]={9,3,7};

 printf ("Elementi del primo vettore: ");
 for (i=0; i<4; i++) printf("%d ",numeri[i]);
 printf("\n");
 printf ("Elementi del secondo vettore: ");
 for (i=0; i<3; i++) printf("%d ",numbers[i]);
 printf("\n");
 if (max_ele(numeri,4) > max_ele(numbers,3)) {
    printf("Il max e' %d ", max_ele(numeri,4))
    printf("ed e' contenuto nel primo vettore");
 }
 else {
    printf("Il max e' %d ", max_ele(numbers,3))
    printf("ed e' contenuto nel secondo vettore");
 }
return 0;
}
```


Funzioni: passaggio per riferimento

La funzione swap e il passaggio di array

Passaggio dei parametri per riferimento (*by reference*)

- Cosa succede se la funzione deve restituire più di un valore alla funzione chiamante?
- Si passano gli indirizzi delle variabili in cui voglio siano contenuti i risultati, in modo che la funzione modifichi realmente QUELLE variabili (e non che ne usi solo i valori per calcolare il risultato)

Passaggio dei parametri per riferimento (*by reference*)

- Esistono variabili che contengono valori che vengono interpretati come indirizzi di memoria: si chiamano *puntatori* perché *puntano* (indirizzano a) una certa cella della memoria centrale.
- Per dichiarare una variabile come puntatore, al momento della dichiarazione si fa precedere il nome da un asterisco.
- Se, in un'espressione, trovo un puntatore *punt* preceduto da un asterisco, l'espressione **punt* rappresenta *il valore* contenuto nella cella di memoria che ha indirizzo uguale al valore di *punt*
- Si ricordi inoltre che l'espressione *&var* restituisce *l'indirizzo* della variabile *var*

Valori		125	18	25	
Indirizzi	….	1251	1252	1253	….

```
char *p;
p=1252;

printf ("%ld", p);      /* stampa 1252 */
printf ("%c" , *p);     /* stampa  18  */
printf( "%c", p[1]);    /* stampa 25 */
```


Uso dei puntatori

```
#include<stdio.h>
int main(){
int dato=5;             /* definizione di intero */
int *indirizzo_dato;    /* definizione di
                           puntatore ad un intero */

indirizzo_dato = &dato; /* assegna l'indirizzo di
                           dato al puntatore */

printf("Il dato vale %d \n ",dato);
                        /*stampera' il numero 5 */
*indirizzo_dato=3;      /* ora dato vale 3 */
printf("Il dato vale %d \n ",dato);
                        /*stampera' il numero 3 */
return 0;
}
```


Esempio di funzione che deve ritornare più valori

- Calcolare il quadrato e il cubo di un dato in una sola funzione
- *Ricorda: se una funzione ha come parametro formale un puntatore, il corrispondente parametro attuale nella chiamata dovrà contenere un indirizzo !!!*


```
#include <stdio.h>

void calcola (float valore, float *quad_val, float *cub_val) {
       *quad_val = valore * valore;
       *cub_val = *quad_val  *  valore;
       }


int main() {
       float  un_dato, quadrato, cubo;
       printf("Introduci un dato");
       scanf("%f", &un_dato);
       calcola (un_dato, &quadrato, &cubo);
       printf("Il quadrato di %f e\':%f \n",un_dato, quadrato);
       printf("Il cubo di %f e\' : %f \n",un_dato, cubo);
       return 0;     }
```

Osservazione

- Importante la sintassi, si rischia di effettuare errori che il compilatore non trova
- Attenzione agli effetti collaterali: occorre valutare che le variazioni introdotte nelle procedure non abbiano ripercussioni indesiderate in altre parti del programma

Esempio: funzione swap

(funzione che commuta il valore fra a e b)

```
void swap (int a, int b)
/*    a e b passati per valore   */
{
int t;
 t = a;  a = b;  b = t;
}
```

Chiamando swap(x,y)
NON SI HA ALCUN EFFETTO su x e y

Esempio: funzione swap

(funzione che commuta il valore fra a e b)

```
void swap (int *a, int *b)
//a e b ora sono dei puntatori
{
 int t;
 t=*a; *a=*b; *b=t;
}
```

Chiamando swap(&x,&y)
si OTTIENE lo scambio desiderato fra x e y.


```
#include <stdio.h>

void scambia(int *x, int *y) {
   int z;
   z = *x;   *x = *y;   *y = z;
}

void swap (int x, int  y) {
      int z;
      z=a;       a=b;        b=z;
}

int main() {
int a=3, b=5;
printf("Prima dello scambio: a=%d, b=%d.\n", a, b); /* a=3 e b=5 */

swap(a, b);
printf("Dopo lo scambio con swap: a=%d, b=%d.\n", a, b);
                                                    /* a=3 e b=5 */

scambia(&a, &b);
printf("Dopo lo scambio con scambia: a=%d, b=%d.\n", a, b);
                                                    /* a=5 e b=3 */

 return 0; }
```

Esempio di
-passaggio per valore
-passaggio per reference
-passaggio di un array
-passaggio di una cella di un array

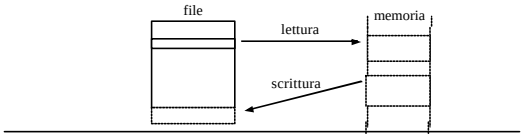

```
#include<stdio.h>
void prova(int a[], int b, int n)
void prova1(int a, int *b);

void prova(int a[], int b, int n)
// a è passato per riferimento, b ed n per valore
{
 int i;
 for (i=1; i<n; i++)
   { a[i]=b; }
 b = a[0];
}

void prova1(int a, int *b)
{
 *b=a;
}
```

```
int main()
{
 int c[3], d;
 c[0]=100; c[1]=15; c[2]=20; d=0;

 printf("Prima: %d,%d,%d,%d\n", c[0],c[1],c[2],d);
 prova(c, d, 3);
 printf("Dopo prova: %d,%d,%d,%d\n", c[0],c[1],c[2],d);
 prova1(c[0], &d);
 printf("Dopo prova1: %d,%d,%d,%d\n",c[0],c[1],c[2],d);
 return 0;
}


Output del programma

Prima: 100,15,20,0
Dopo prova: 100,0,0,0
Dopo prova1: 100,0,0,100
```

I File

I File

- normalmente con la parola *file* (o *archivio*) si intende un insieme di informazioni logicamente correlate, memorizzate su memoria di massa
- le operazioni tipiche che si possono eseguire su di un file sono:
  - *lettura* di dati in memoria
  - *scrittura* (aggiornamento o aggiunta) di dati dalla memoria



I File

- un file è una struttura *dinamica*, in quanto è possibile aggiungere dei nuovi dati al suo interno, incrementando così la sua dimensione
- per il C, un file non è altro che una sequenza di byte
- esistono due tipi di file in C
  - **file di testo**
    - sono strutturati in linee di testo (tipicamente in codice ASCII)
    - il C identifica la fine di una linea, con il carattere '\n'
    - l'effettivo contenuto del file su disco, dipende dal modo in cui il sistema operativo consente di realizzare i file di testo
  - **file binari**
    - i byte che vengono letti e scritti, corrispondono all'effettivo contenuto del file: non viene effettuata alcuna conversione
    - anche un file di testo può essere elaborato come file binario: in tal caso si opera sull'effettivo contenuto in byte del file

I File

- più in generale, il C considera file un qualunque dispositivo di I/O da cui possono essere letti o su cui possono essere scritti dei byte di informazioni
  - Es. tastiera, monitor, stampante, file su disco, porta seriale …
- esistono in particolare tre file standard utilizzabili direttamente da parte di un qualunque programma C:
  - ***stdin*** (standard input): è il dispositivo standard di input dei dati, normalmente rappresentato dal terminale; viene usato dalle funzioni scanf(), getchar(), gets(), …
  - ***stdout*** (standard output): è il dispositivo standard di output dei dati, normalmente rappresentato dal monitor del terminale; viene usato dalle funzioni printf(), putchar(), puts(), …
  - ***stderr*** (standard error): è il dispositivo standard di visualizzazione dei messaggi di errore, normalmente rappresentato dal monitor del terminale

I File

- quando si desidera elaborare un file occorre dichiarare un puntatore ad una struttura di dati apposita, di nome *FILE* e definita nel file stdio.h

  Es. `FILE* mio_file;`

- la struttura serve per memorizzare informazioni interne indispensabili per l'elaborazione successiva del file stesso; chi scrive il programma non è tenuto a conoscerne l'effettivo contenuto
- il puntatore serve, all'interno del programma, per fare riferimento al file interessato, nelle successive operazioni
- tale struttura non deve essere dichiarata per i tre file standard

I File

- la prima operazione che deve essere effettuata su di un file è la sua ***apertura***

  `FILE* fopen(char* nomefile, char* modalità);`

  - ***nomefile*** rappresenta il nome effettivo del file
  - ***modalità*** specifica il modo in cui il file verrà aperto; condiziona le successive operazioni sul file stesso

- la funzione fopen() crea una struttura di dati di tipo FILE, restituisce come risultato il ***puntatore*** a tale struttura; in caso di errore (file inesistente, ...) restituisce NULL
- i tre file standard vengono aperti automaticamente

I File

"r"	apre un file di testo esistente consente operazioni di lettura si posiziona all'inizio del file
"w"	crea un nuovo file di testo; consente operazioni di scrittura
"a"	apre un file di testo esistente si posiziona alla fine del file consente operazioni di scrittura (aggiunta di linee alla fine del file)
"r+"	apre un file di testo esistente consente operazioni di lettura e di scrittura (in un qualunque punto, con sovrascrittura dei byte preesistenti) si posiziona all'inizio del file
"w+"	crea un nuovo file di testo consente operazioni di lettura e di scrittura
"a+"	apre un file di testo esistente si posiziona alla fine del file consente operazioni di scrittura

Inserendo la lettera "b" si ottengono le analoghe modalità per i file binari

I File

- Alla fine dell'elaborazione di un file occorre effettuare l'operazione di *chiusura*

```
int fclose (FILE* puntatore)
```

  - libera la struttura di dati abbinata al file
  - in caso di scrittura, garantisce che tutti i dati scritti sul file, siano effettivamente trasferiti su disco
  - ritorna 0 se conclusa correttamente
  - comunque, alla terminazione di un programma C, il programma stesso provvede **a chiudere tutti gli eventuali file ancora aperti** (ma non ne garantisce la corretta scrittura)

File di Testo

- per leggere un carattere da un file di testo si utilizza la funzione:

```
int fgetc (FILE* puntatore)
```

  - legge un carattere dal file ed avanza sul carattere successivo
  - restituisce il carattere letto sotto forma di numero int (con i bit oltre il primo byte, tutti uguali a zero)
  - restituisce il carattere '\n' quando viene incontrata la fine di una riga
  - restituisce un valore apposito, rappresentato dalla costante EOF (definita in stdio.h, solitamente con il valore -1), per indicare che si è raggiunta la fine del file

Esempio

```
/* legge i caratteri presenti in un file di testo e li invia al monitor*/
#include <stdio.h>
int main(){
  char c,nome[100];
  FILE* f;
  printf ("Specificare il nome del file:");
  scanf ("%s", nome);
  if ((f=fopen(nome, "r")) == NULL)
     printf ("Errore apertura file\n");
  else{
     while((c=fgetc(f))!=EOF)
          putchar (c);
     fclose(f);
  }
  return 0;
}
```

equivale a:
```
f=fopen(nome, "r");
if(f == NULL)
…
```

equivale a:
```
c=fgetc(f);
while(c!= EOF) {
  putchar (c);
  c=fgetc(f);
}
```


File di Testo

- per scrivere un carattere in un file (di testo o binario), si utilizza la funzione:

```
int fputc(int carattere, FILE* puntatore);
```

  - scrive un carattere nella posizione corrente del file
  - ritorna il carattere scritto, in caso di successo, EOF in caso di errore
  - qualora si scriva il carattere di codice '\n' in un file di testo, si ha l'effetto di chiudere la linea corrente del file

Esempio

```
/*aggiunge una riga alla fine di un file di testo, leggendo i caratteri da tastiera*/
#include <stdio.h>
int main(){
  char c,nome[100];
  FILE* f;
  printf ("Specificare il nome del file:");
  scanf ("%s", nome);
  getchar(); /* elimina '\n' dal buffer della tastiera */
  if ((f=fopen(nome, "a")) == NULL)
     printf ("Errore apertura file\n");
  else{
     do{
       c=getchar();
       fputc(c, f);
     }while (c!='\n');
     fclose(f);
  }
  return 0;
}
```


File di Testo

- Per leggere una stringa da un file (di testo o binario), avanzando del numero di caratteri corrispondente, si utilizza la funzione:

```
char* fgets(char* stringa,int lunghezza,FILE* puntatore);
```

  - occorre specificare la stringa destinata a contenere i caratteri letti ed il numero massimo di caratteri che possono essere letti (compresi i caratteri '\n' e '\0')
  - alla fine della stringa letta, viene memorizzato anche il carattere '\n', oltre al carattere di fine stringa '\0'
  - la funzione termina quando viene letto il carattere '\n' (effettivo o convertito) oppure quando viene esaurita la lunghezza specificata
  - ritorna NULL in caso di fine file o in caso di errore; (ritorna il puntatore alla stringa fornita come parametro, in caso di successo)

Esempio

```
/*legge le righe di un file di testo e le visualizza*/
#include <stdio.h>
int main() {
  char nome[100],riga[100];
  FILE *f;
  printf ("Specificare il nome del file:");
  scanf ("%s", nome);
  getchar(); /* elimina '\n' dal buffer della tastiera */
  if ((f=fopen(nome, "r")) == NULL)
     printf ("Errore apertura file\n");
  else{
    while (fgets(riga,100,f)!=NULL)
       printf("%s",riga);
   fclose(f);
}
  return 0;}
```

Il carattere '\n' è già presente nella stringa letta

File di Testo

- per scrivere una stringa in un file (di testo o binario), si utilizza la funzione:

```
int fputs(const char* stringa, FILE* puntatore);
```

  - non aggiunge automaticamente il carattere '\n' alla fine della linea
  - ritorna EOF in caso di errore

Esempio

```
/* legge delle stringhe e le scrive in un file di testo nuovo;
   terminaquando viene inserita una riga vuota */
#include <stdio.h>
#include <string.h>
int main() {
  char nome[100],riga[100];
  FILE* f;
  printf ("Specificare il nome del file:");
  scanf ("%s", nome);
  getchar(); /* elimina '\n' dal buffer della tastiera */
  if ((f=fopen(nome, "w")) == NULL)
     printf ("Errore apertura file\n");
  else {
      do {
        gets(riga);
        if (riga[0]!='\0') {
            strcat(riga, "\n");
            fputs(riga,f);
        }
     }while (riga[0]!='\0');
    fclose(f);
  }
  return 0;}
```


fscanf e fprintf

- Corrispondono a printf e scanf (che operano rispettivamente sui file stdout e stdin) solo che consentono di specificare il file su cui scrivere o da cui leggere
- Esempi
  - fscanf(miofile, "%d",&intero);
  - fprintf(miofile, "%d\n",intero);

  printf("%d\n",intero); equivale a
  fprintf(stdout, "%d\n",intero);


```
/* Legge due temperature in gradi Celsius da un file, le converte
   in gradi Fahrenheit e le salva sul secondo file*/

#include <stdio.h>

int main() {
  float c1, c2, f1, f2;
  char nomein[20], nomeout[20];
  FILE* filein, *fileout;

  printf ("Specificare il nome del file di input:");
  scanf ("%s", nome);
  getchar();      /* elimina '\n' dal buffer della tastiera */

  if ((filein=fopen(nomein, "r")) == NULL)
     printf ("Errore apertura file di input\n");
  else
/* continua . . . */
```



```
{
  printf ("Specificare il nome del file di output:");
  scanf ("%s", nomeout);
  getchar(); /* elimina '\n' dal buffer della tastiera */
     if ((fileout=fopen(nomeout, "w")) == NULL)
         printf ("Errore apertura file di output\n");
     else {
            fscanf(filein, "%d", &c1);
            fscanf(filein, "%d", &c2);

            /* mettiamo esplicitamente lo spazio di separazione */
            f1=32+c1*9/5;
            f2=32+c2*9/5;
            fprintf(fileout, "%d ", f1);
            fprintf(fileout, "%d", f2);
            /* fra il 1 e 2 numero con uno spazio dopo il %d */
            fclose(fileout);
          }
  fclose(filein);
}
return 0;}
```

Esercizio

- Scrivere una funzione
  int doppiasomma(int *punt1, int *punt2) che raddoppia il valore del contenuto delle locazioni puntate da punt1 e punt2 e ne restituisce la somma, dopo il raddoppio.
- Scrivere una funzione
  int double_sum(int a, int b) che raddoppia i valori passati in a e in b e ne restituisce la somma, dopo il raddoppio.
- Scrivere un programma che definisce due variabili intere x e y, assegna loro due valori e successivamente chiama la funzione doppiasomma e double_sum utilizzando come parametri le due variabili x e y, passate, nel primo caso, per riferimento e, nel secondo, per valore, stampando il valore di x e di y dopo ogni chiamata.
- Spiegare il diverso comportamento delle due funzioni.

Soluzione (funzioni)

```
int doppiasomma(int *punt1, int *punt2)
{
 *punt1= *punt1  *  2;
 *punt2= *punt2  *  2;
return(*punt1 + *punt2);
}

int double_sum( int a, int b)
 {
a = a*2;
b = b*2;
return (a+b);
}
```


Soluzione (main)

```
int main()
{
int x=2,y=5,ris;
ris = doppiasomma(&x,&y);
printf("ris = %d, x=%d, y=%d", ris,x,y);
/* adesso x vale 4 e y vale 10 */
ris = double_sum(x,y);
printf("ris = %d, x=%d, y=%d", ris,x,y);
/*adesso x vale ancora 4 e y vale ancora 10 */
return (0);
}
```