Esempio di funzione passata come parametro, opzione 1: si specifica solo il tipo del valore restituito

```
int som(int (f) (), int m, int n);
/* calcola la somma f(m) +...f(n)
*postc: se m<=n restituisce f(m) +...+f(n), se n<m restituisce 0*/

int som(int (f)(), int m, int n)
/* calcola la somma f(m) +...f(n)
* postc: se m<=n restituisce f(m) +...+f(n), se n<m restituisce 0*/
{int i;
int somma =0;
for (i=m; i <= n;i++)
            somma  += f(i);
return somma;
}
```

Esempio di funzione passata come parametro, opzione 2: anche il tipo del parametro in input è specificato

```
int som(int (f) (int), int m, int n);
/* calcola la somma f(m) +...f(n)
*postc: se m<=n restituisce f(m) +...+f(n), se n<m restituisce 0*/

int som(int(f)(int), int m, int n)
/* calcola la somma f(m) +...f(n)
* postc: se m<=n restituisce f(m) +...+f(n), se n<m restituisce 0*/
{int i;
int somma =0;
for (i=m; i <= n;i++)
            somma  += f(i);
return somma;
}
```

Esempio di funzione passata come parametro, opzione 3: si evidenzia il puntatore allla funzione

```
int som(int (*f) (), int m, int n);
/* calcola la somma f(m) +...f(n)
*postc: se m<=n restituisce f(m) +...+f(n), se n<m restituisce 0*/

int som(int (*f)(), int m, int n)
/* calcola la somma f(m) +...f(n)
* postc: se m<=n restituisce f(m) +...+f(n), se n<m restituisce 0*/
{int i;
int somma =0;
for (i=m; i <= n;i++)
            somma  += f(i);
return somma;
}
```

Esempio di funzione passata come parametro

```
int pot2(int x);
/* calcola la x-sima potenza di 2
* prec: x >=0
*postc: se x>=0 restituisce 2 elevato a x, 1 altrimenti  */

int pot2(int x)
/* calcola la x-sima potenza di 2
* prec: x >=0
*postc: se x>=0 restituisce 2 elevato a x, 1 altrimenti  */
{int pot = 1, i=0;
 assert(x>=0);
 while (++i <=x)
            pot *= 2;
 return pot; }

int ident(int x);

int ident(int x)
{return x;}
```

Esempio di funzione passata come parametro

```
int Gauss(int x)
/*restituisce la somma dei primi  x interi
*prec x>=0;
*postc: restituisce x*(x+1)/2 */
{assert(x>=0);
return x*(x+1)/2;}

int somPot2(int x)
/*restituisce la somma delle potenze di 2, da 0 a x.
*prec x>=0;
*postc: restituisce (2 elevato x+1) -1 */
{assert(x>=0);
return pot2(x+1) -1;}
```

Esempio di funzione passata come parametro

```
int main()
{int i, res, num, test;
printf("dammi il numero di test che vuoi fare \n ?\n");
scanf("%d",&test);
for (i=0;i<test;i++)
    {printf("dammi il numero di elementi per il  test numero %d \n ?\n",i);
      scanf("%d",&num);
      if ((res = som(ident,0,num)) == Gauss(num))
      printf("la somma dei primi %d interi è %d\n",num,res);
      if ((res = som(pot2,0,num)) == somPot2(num))
      printf("la somma delle prime %d potenze è %d\n",num,res);
      }
return 0;}
```

La dichiarazione

```
int (g) ();
```

introduce un puntatore costante g, mentre

```
int (*h) ();
```

dichiara h come puntatore a una funzione e quindi

g = pot2; non è ammissibile perchè g non può cambiare valore, mentre

h = pot2; assegna ad h l' indirizzo di pot2.

ma

h = pot2(); non va perchè pot2() restituisce un intero, non un puntatore

```
int (*h) ();  /* h è un puntatore a una funzione che restituisce un intero
*/

int main()
{int i, res, num, test;
h = ident;
printf("dammi il numero di elementi per il  test \n ?\n");
scanf("%d",&num);
if ((res = som(h,0,num)) == Gauss(num))
            printf("la somma dei primi %d interi è %d\n",num,res);
h = pot2;
if ((res = som(h,0,num)) == somPot2(num))
            printf("la somma delle prime %d potenze è %d\n",num,res);
return 0;}
```

Puntatori a funzione: indipendenza dal numero e dal tipo degli argomenti

```
int (*g) (); /* g è un puntatore a una funzione che restituisce un intero  */

int main()
{int i,res, num, test;
g = som;
printf("dammi il numero di elementi per il  test \n ?\n");
scanf("%d",&num);
if ((res = g(ident,0,num)) == Gauss(num))
      printf("la somma dei primi %d interi è %d\n",num,res);
g = pot2;
if ((res = som(g,0,num)) == somPot2(num))
       printf("la somma delle prime %d potenze è %d\n",num,res);
return 0;}
```

Però quando la funzione è passata come parametro basta il nome della funzione così come quando la si valuta

g(x) == (*g) (x) == (g) (x) == (****g)(x)**

infatti prima il compilatore prima risolve le "stelline" e solo quando arriva al nome della funzione (che è a sua volta un puntatore) valuta l'espressione.

Calcolo in millisecondi del tempo preso dal codice rappresentato da ...

```
clock_t start, end;
double millisec;
time_t app;

{start = clock();
 ...
end = clock();
millisec = (end-start)/(CLOCKS_PER_SEC /(double) 1000.0);
```

NOTA : la precisione in millisecondi non è necessariamente raggiunta da clock(). Su molte piattaforme la precisione non è superiore a 50 millisecondi (così se la differenza tra due successive chiamate di clock è minore di 50, il risultato è 0). C'è anche sempre il rischio di overflow, che dipende dalla grandezza del tipo clock_t e dal valore di CLOCKS_PER_SEC.

clock();

Prototipo in <time.h>: `clock_t clock(void);`

Parametri: La funzione non ha parametri.

Valore restituito: La funzione restituisce un valore di tipo `clock_t` che rappresenta la migliore approssimazione del tempo utilizzato dal programma fino al punto della chiamata.

Nota: E' usata per ottenere valori di tipo clock_t che possono essere usati per calcolare il tempo trascorso durante l'esecuzione di un programma. Per ottenerlo in secondi basta dividere il valore ottenuto per `CLOCKS_PER_SEC`, una costante definita in `time.h`, che contiene il numero per secondo del valore restituito da clock

Attenzione!

`clock_t`, è un tipo aritmetico dipendente dal sistema ed è definito in `time.h`.

time e ctime

Prototipo in <time.h>:

```
time_t time(time_t *timer);
char *ctime(const time_t *timer);
```

Parametri: Un puntatore (indirizzo) a una variabile di tipo time_t, che è il tipo, dipendente dal sistema, usato per rappresentare date e tempi.

Valore restituito da time: La funzione restituisce il numero di secondi, a partire dal 1° gennaio 1970
Valore restituito da ctime: La funzione converte la data in input in un array di caratteri

Nota: Se timer != NULL, *timer assume il valore restituito da time.

Esempio di uso:

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

int main(void)
{time_t systime;
systime = time(NULL);
puts(ctime(&systime));
return 0;
}
```

Output:
Tue Apr 30 16:10:08 2002