Funzioni come parametri

- Obiettivi:
  - Riprendere il modello computazionale alla base del linguaggio C
  - Presentare e sperimentare le funzioni come parametri di funzione

LINGUAGGI DI ALTO LIVELLO

Si basano su una *macchina virtuale* le cui "mosse" non sono quelle della macchina hardware

label	op. code	operands / arguments	comments
	BC	4,MINUS	
	MVI	SIGN,X'00'	STORE PLUS
	BC	15,NEXT	
MINUS	MVI	SIGN,X'80'	STORE MINUS
	LPR	0,0	COMPLEMENT TO POSITIVE
NEXT	LA	3,32	SUBTRACT SCALE FACTOR FROM 32
	SR	3,1	
	SR	2,2	
	D	2,K4	DIVIDE BY 4
	LNR	2,2	COMPLEMENT REMAINDER TO NEGATIVE
	BC	8,NONE	NO REMAINDER
	A	2,K4	SUBTRACT REMAINDER FROM 4
	LA	3,1(3)	ROUND UP EXPONENT
NONE	SRDL	0,8(2)	ALIGN CONSTANT
	STM	0,1,STORE	
	LA	3,64(3)	CREATE EXPONENT BY ADDING 64
	STC	3,STORE	MOVE HEX EXPONENT TO MANTISSA
	OC	STORE(1),SIGN	OR-IN SIGN
	SWR	0,0	
	AD	0,STORE	LOAD AND NORMALIZE
STORE	DS	1D	
K4	DC	F'4'	
SIGN	DS	1C	

Brano di un programma scritto in assembler (IBM 360)

Quali astrazioni?

- Sui dati: da indirizzi fisici a ***nomi simbolici*** per le variabili (Assembler)
- Sul controllo (***programmazione strutturata***, Pascal, C):
  - sequenza (;),
  - blocco,
  - **`if else`**, **`while do`**, etc.

```
{ int p=1;
  for (i=1 ; i <= n ; ++i)   p = p*x;
  . . .
 }
```

Astrazioni funzionali

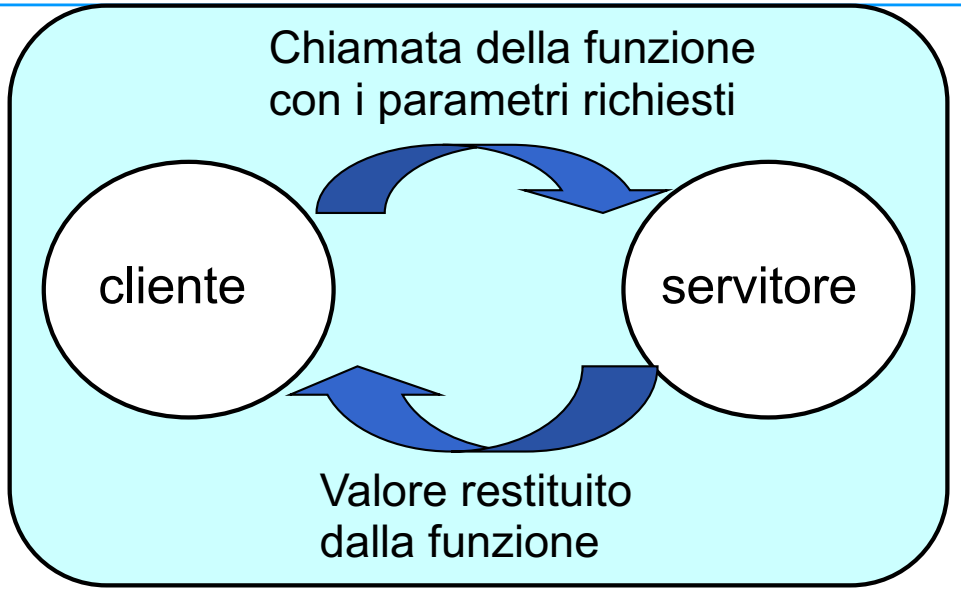
- Sulle operazioni (***funzioni***):

```
long potenza(int x, int n) /* par. formali */
{ int i;
  long p = 1;                /* var. locali */
  for (i=1 ; i <= n ; ++i)   p *= x;
  return p;
}
```

```
long Y;
Y=potenza(X,10);
```

INTERFACCIA DI UNA FUNZIONE

- L'interfaccia (o firma o ***signature***) di una funzione comprende
  - nome della funzione
  - lista dei parametri
  - tipo del valore da essa denotato



- Definisce *il contratto di servizio* fra cliente e servitore.
- Cliente e servitore comunicano quindi mediante

  – i *parametri* trasmessi dal cliente al servitore all'atto della chiamata (direzione: dal cliente al servitore; se trasferimento per indirizzo, bidirezionalità delle informazioni)

  – il *valore restituito* dal servitore al cliente direzione: dal servitore al cliente)

ESEMPIO

Parametri Formali

```
long potenza(int x, int n)
{ int i;
  long p = 1;/* var. locali*/
  for (i=1 ; i <= n ; ++i)
      p *= x;
  return p;     }
```

SERVITORE

definizione della funzione

```
   main()
   {    long Y;
        Y=potenza(X,10);
   }
```

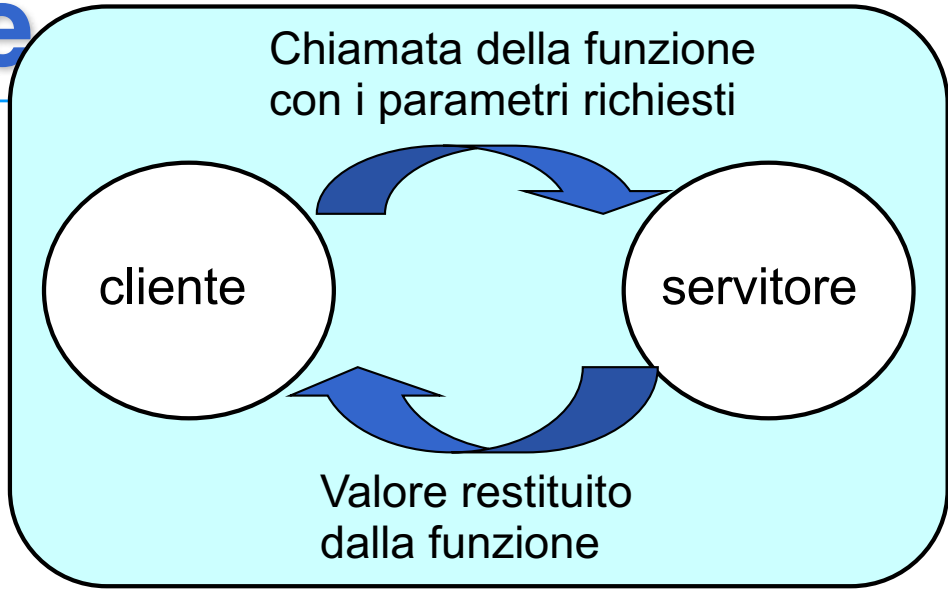
CLIENTE

chiamata della funzione

Parametri Attuali

Parametri funzione

- Tra i ***parametri formali*** di una funzione F ci possono essere
  - Funzioni
- F è un ***funzionale*** (funzione che opera su funzioni)



- In C, l'***identificatore*** di una funzione è il ***puntatore*** al suo codice
- ***Parametro attuale***, nome di una specifica funzione (di libreria o *user-defined*)
- ***Parametro formale***, puntatore a codice di una funzione con un certo tipo restituito e tipo dei parametri formali

Parametri funzione

- Per avere parametri procedura o funzione, occorre specificare un ***parametro formale*** di tipo funzione, come segue (*signature* della funzione F1 che ha la funzione F2 come parametro formale):

```
tipo1 F1 (tipo2 (*F2)(list-pf-F2), rest-param-F1)
```

- Nel corpo di F1 potrà esserci la chiamata:

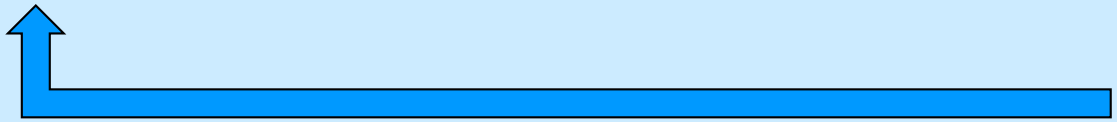
```
k=(*F2)(...)    /* con parametri attuali opportuni */
```

- Procedure e funzioni possono apparire non solo come ***parametri*** di altre procedure e funzioni, ma possono anche essere restituite (attraverso un puntatore) come ***risultato*** di una funzione

Parametri funzione: esempio

- Un ***parametro formale*** funzione è specificato indicando un ***nome*** (puntatore), la ***lista dei parametri formali*** ed il ***tipo di risultato***.

```
double sommaquadratif(
      double (*f)(double par), int m, int n)
{ int k;
  double somma;
  somma = 0;
  for (k=m; k <= n; k++)
     somma = somma + (*f)(k) * (*f)(k);
  return somma; }
```

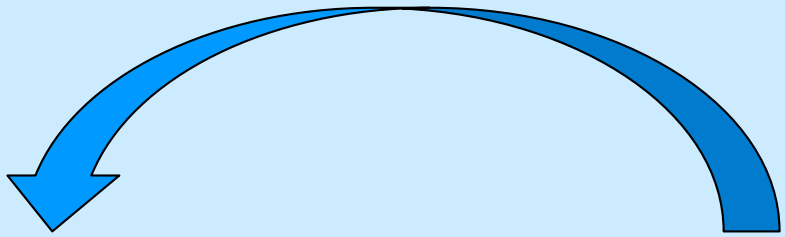


Parametri funzione: esempio

- Il ***parametro attuale*** con cui si invoca F1 deve essere un identificatore di funzione con la stessa descrizione dei parametri e lo stesso tipo di risultato di F2

```
#include <math.h>
/* double sin (double);    libreria math.h*/

void main (void)
{ printf (" Seni %.7f\n",
          sommaquadratif (sin, 2, 13));  }
```



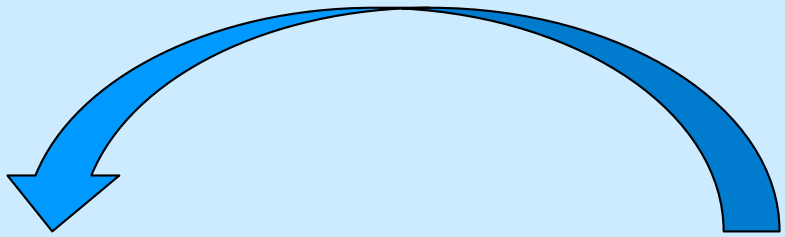
- La funzione **`sin`** della libreria **`math.h`** ha l'interfaccia del parametro funzione del funzionale **`sommaquadratif`**

Parametri funzione: esempio

- Il ***parametro attuale*** con cui si invoca F1 deve essere un identificatore di funzione con la stessa descrizione dei parametri e lo stesso tipo di risultato di F2

```
#include <math.h>
/* double sin (double);    libreria math.h*/

void main (void)
{ printf (" Seni %.7f\n",
          sommaquadratif (my_fun, 2, 13));  }
```



- oppure posso definire io una funzione

Esempio 2

```
#include <math.h>
/* double sin (double);   libreria math.h*/
double my_fun (double x) { return 1.0 / x;} /* reciproco */

double sommaquadratif (double (*f)(double par),int m,int n)
{ int k;
  double somma=0;
  for (k=m; k <= n; k++) somma = somma + pow((* f)(k),2);
  return somma;  }

double sommacubif ( double (*f)(double par),int m,int n)
{ int k;   double cubo=0;
  for (k=m; k <= n; k++)
     cubo=cubo + pow((* f)(k), 3);
  return cubo;  }
```

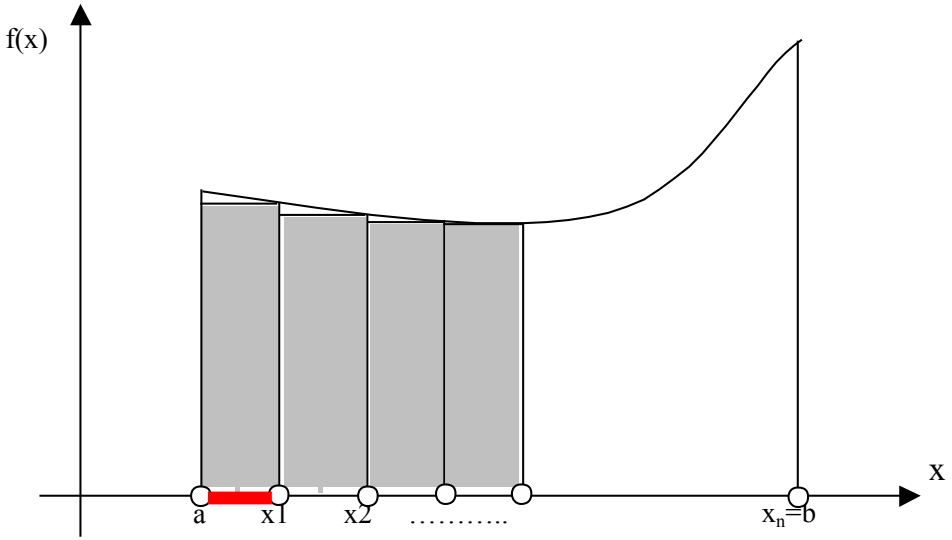
Esempio 2 (cont.)

```
void main (void)
{printf (" Inversi %.7f\n",
        sommaquadratif (my_fun,1, 10000));
 printf (" Seni %.7f\n",
         sommaquadratif (sin, 2, 13));
 printf (" Inversi %.7f\n",
         sommacubif (my_fun, 1, 10000));
 printf (" Seni %.7f\n",
         sommacubif (sin, 2, 13));}
```

- Il codice diventa parametrico nella funzione definita come parametro, e può essere invocato con diverse funzioni passate come parametro attuale (nell'esempio, `my_fun` e `sin`)

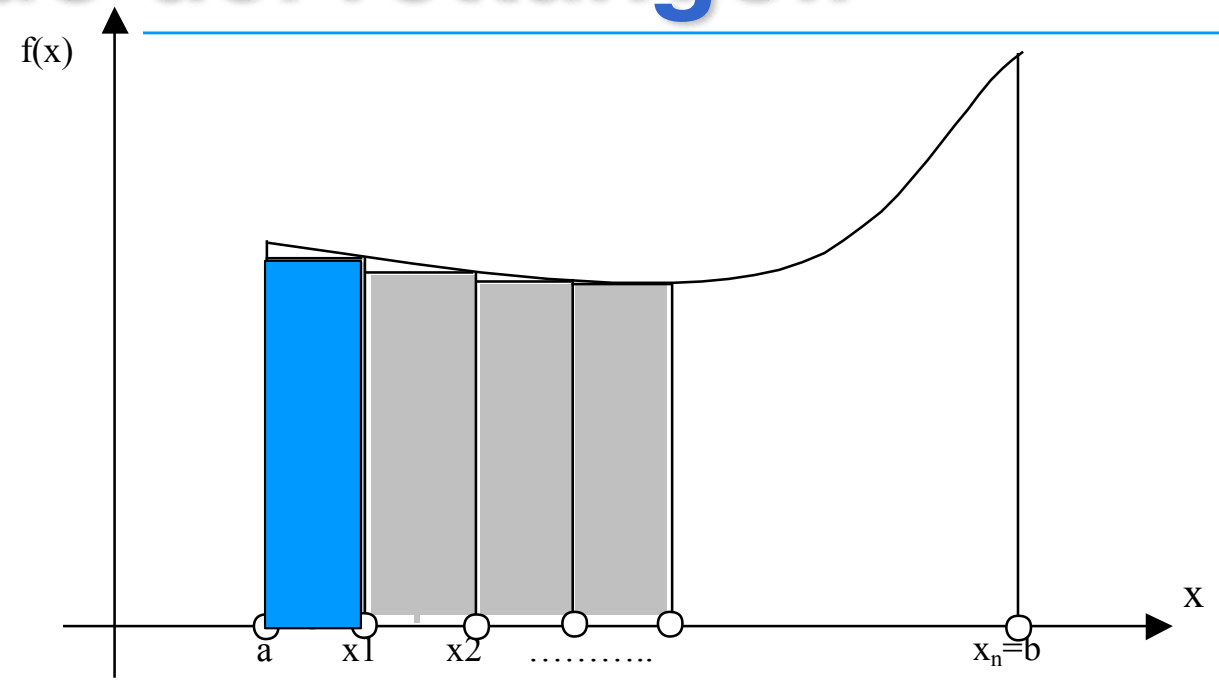
Esercizio (in lab): integrazione

- Si realizzi un programma che calcola l'integrale (in modo approssimato) di una ***funzione reale di una variabile reale*** in un intervallo dato [a,b] con il ***metodo dei rettangoli***.



- Data la funzione integranda f, si suddivide l'intervallo di integrazione [a,b] in N sotto-intervalli di uguale ampiezza h. Il valore di **h** è quindi dato da **(b-a)/N**.

Metodo dei rettangoli



- Il contributo I_i di ciascun sotto-intervallo nel calcolo dell'integrale è quindi: $I_i = h*f(x_i)$
- L'integrale I si calcola quindi come ***sommatoria*** dei contributi di tutti i sotto-intervalli:

$$I = \Sigma_{i=1..N} I_i = \Sigma_{i=1..N} h*f(x_i) = h*\Sigma_{i=1..N} f(x_i)$$

Soluzione: to do

```
float rettangoli(float (*func)(float),
                 float a, float b, int n)
/* func funzione integranda; a,b estremi
   intervallo; n numero sotto-intervalli */
{  ...
  }
```

- Il main deve leggere da input gli estremi (sia 1000 il numero di sotto-intervalli) e chiamando questa funzione calcolare l'integrale di x^2 e x^3.