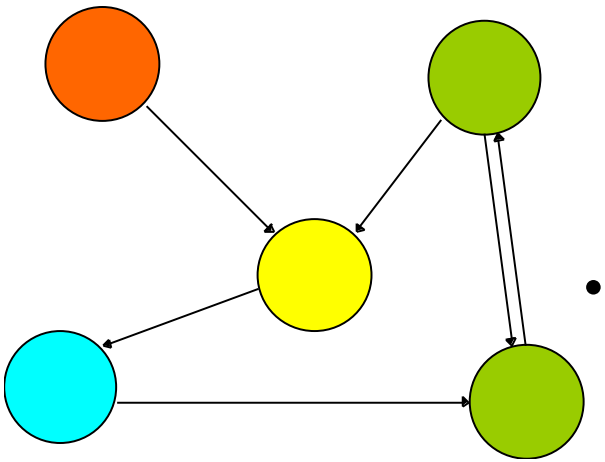
Funzioni e procedure

Le funzioni di libreria

```
#include <stdio.h>
main()
    {
     int a, b, somma;
       printf("Immettere due numeri: ");
       scanf("%d %d", &a, &b);
       somma=a+b;
       printf("Somma: %d\n", somma);
    }
```

- Le funzioni di libreria `printf(..)` e `scanf(..)` svolgono un servizio per la funzione `main()`
  - Svolgono il ruolo di servitori: SERVER
- La funzione `main()` si avvale dei servizi offerti da `printf(..)` e `scanf (...)`
  - Svolge il ruolo di cliente: CLIENT
- `printf(..)` e `scanf(..)` sono *dichiarate* in `stdio.h`
  - Con la direttiva di precompilazione `#include <stdio.h>` si include nel programma il contenuto della libreria di sistema `stdio.h`

Architetture multi-componente



- In generale, una applicazione è organizzata come insieme di **COMPONENTI SOFTWARE** che cooperano per raggiungere un fine comune.
- Nello specifico, un **programma** può essere organizzato come un insieme di **FUNZIONI** che cooperano per risolvere un problema

- Un **PROGRAMMA IN C** contiene sempre almeno una funzione: la funzione `main()`
- Oltre al `main()` e alle funzioni di libreria, è possibile aggiungere altre funzioni

Funzioni e procedure

Quali MODELLI *per costruire un'architettura multi-componente?*

- ✓ Il modello cliente-servitore

Quali ASTRAZIONI*, quali modelli per costruire i singoli componenti?*

Quali PROTOCOLLI *di comunicazione fra componenti?*

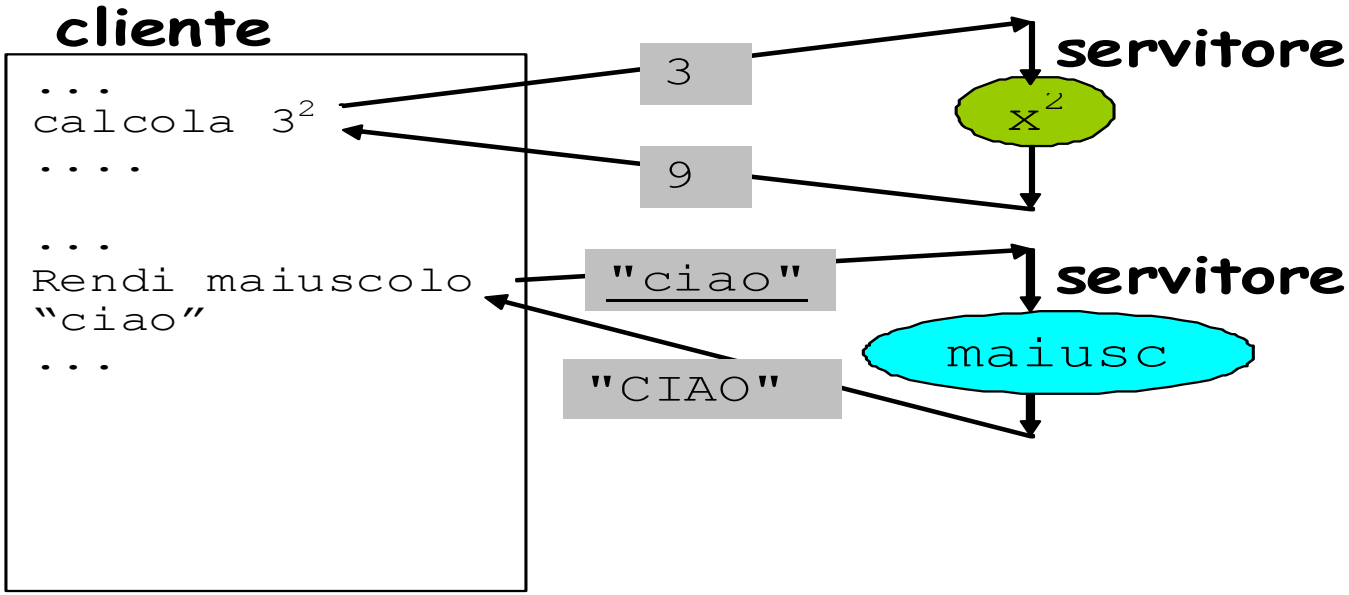
- ✓ Definizione e uso di funzioni e procedure
- ✓ Dichiarazione e definizione
- ✓ Passaggio per valore e passaggio per riferimento
- ✓ Parametri formali e parametri attuali

Il modello cliente/servitore

Un modello molto generale per impostare un'architettura a componenti è il cosiddetto **MODELLO CLIENTE/ SERVITORE:**

Ogni **COMPONENTE**

- *fornisce* servizi ad altri componenti: **E' UN SERVITORE per essi**
- e *usa i servizi* forniti da altri componenti: **E' UN CLIENTE di servizi altrui**



Comunicazione tra cliente e servitore

- Per chiedere al servitore di svolgere un servizio occorre
  - ***chiamarlo per nome***
  - ***passargli le necessarie informazioni***
- All'atto della CHIAMATA, l'esecuzione del cliente viene ***sospesa*** e il ***controllo*** passa al servitore.
- Cliente e servitore comunicano mediante:
  - i PARAMETRI trasmessi dal cliente all'atto della chiamata del servitore
  - l'eventuale VALORE RESTITUITO dal servitore al cliente

Parametri

PARAMETRI FORMALI

Sono specificati nella **DICHIARAZIONE** del servitore e indicano *cosa il servitore si aspetta dal cliente*

ESEMPI

- Il componente che calcola il quadrato di un numero si aspetta un valore di tipo intero
- Il componente che trasforma in maiuscolo un testo si aspetta una stringa

PARAMETRI ATTUALI

Sono trasmessi dal cliente all'atto della chiamata e devono corrispondere ai parametri formali (in numero, posizione e tipo)

ESEMPI

- Il cliente passa il valore 3 al servitore che calcola il quadrato di un numero
- Il cliente passa il valore "ciao" al servitore che trasforma in maiuscolo

Funzioni e Procedure

FUNZIONE

Componente software che cattura l'astrazione matematica di funzione:

$f : A \times B \times ... \times Q \rightarrow S$

molti possibili ingressi e una sola uscita

PROCEDURA

Componente software che cattura l'astrazione di azione complessa su determinati oggetti

- Le astrazioni di *funzione* e *procedura* sono presenti in tutti i linguaggi di programmazione di alto livello, dove consentono di creare delle *unità di programma*, dotate di **NOME**, che
  - incapsulano le istruzioni
  - realizzano un certo servizio
- I vantaggi derivanti dall'uso di procedure e funzioni sono:
  - riutilizzo di codice
  - migliore leggibilità
  - sviluppo top-down

Uso di funzioni e procedure

Uso di Funzione

È un'astrazione del concetto di *operatore* (funzione o predicato) che calcola un valore

→ può comparire ovunque una espressione accetti un valore del tipo restituito dalla funzione

Uso di Procedura

È un'astrazione della nozione di istruzione

→ può comparire ovunque possa comparire un'istruzione

Le **LIBRERIE STANDARD** mettono a disposizione funzioni e procedure utilizzabili all'interno dei programmi che includono il corrispondente *header* o *file d'intestazione* (ad esempio `stdio.h`)!

Esempi di chiamate di funzione

```
ris = potenza(3) + fabs(x);
for (i=0;i<potenza(N);i++)
for (i=0;!èPotenza(i,N);i++)
int i=scanf("%d %d", &a, &b);
```

Esempio di chiamata di procedura

```
stampa_menu();
```

Dichiarazione

La DICHIARAZIONE di una funzione o procedura costituisce la *specifica del protocollo di comunicazione* fra cliente e servitore.

Come tale, la dichiarazione precisa:

- il nome del componente
- tipo dei parametri di ingresso (event. il nome)
- tipo del valore di uscita

ESEMPIO

```
typedef int Bool;
int potenza(int);
Bool èPotenza(int,int);
```

Definizione

La **DEFINIZIONE** di una funzione o procedura costituisce la *effettiva realizzazione* del componente.

Come tale, la definizione precisa:

- la dichiarazione con i nomi dei **PARAMETRI FORMALI**
- il blocco di istruzioni che realizza l'algoritmo richiesto

ESEMPIO

```
int potenza(int n)
{int p;
 p=n*n;
 return p;}
Bool èPotenza(int n, int pot)
{return pot==n*n;}
```

Definizione di funzione in C

Una **definizione di funzione** è costituita da una *intestazione* e da un *blocco*:

<def-funzione> ::= <intestazione-funzione> **<blocco>**

<intestazione-funzione> ::=
<nomeTipo> <nomeFunzione> (<lista-parametri>)

<lista-parametri> ::=
void | <dich-parametro> { , <dich-parametro> }

<dich-parametro> ::= <nomeTipo> <identificatore>

L'**intestazione** specifica nell'ordine:

- il tipo del risultato
- il nome della funzione
- la lista dei parametri formali
  - `void` se vuota
  - una sequenza di dichiarazioni di parametri, se non vuota
- di ogni parametro si precisano, nell'ordine, *tipo* e *nome.*

Esempio

Scrivere una funzione che calcoli il massimo fra due interi.

CODIFICA

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

NOTE

L'istruzione **`return`** [<espressione>]

- viene usata per *restituire il controllo al chiamante* e nel contempo *restituirgli il valore calcolato* (risultato)
  → eventuali istruzioni successive non saranno eseguite!

  quindi la funzione max poteva essere scritta anche così:

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

- l' <espressione> che segue **`return`** deve denotare un valore *dello stesso tipo del risultato* dichiarato per la funzione

Un programma completo

Dati quattro valori interi, scrivere un programma che stampi la radice quadrata del valore più grande usando la funzione `max`.

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

int max (int, int);

int main() {
int a,b,c,d,m;

printf("Inserisci i quattro valori: ");
scanf("%d %d %d %d",&a,&b,&c,&d);
m=max(max(a,b),max(c,d));
printf("Radice di %d: %f",m,sqrt(m));}

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

Esempio: Modello a tempo di esecuzione

I parametri `a` e `b`:

- sono ***creati*** al momento della chiamata della funzione
- sono ***inizializzati*** con i valori dei corrispondenti parametri attuali trasmessi dal cliente
- ***vivono*** per tutto il tempo in cui la funzione è in esecuzione
- sono ***distrutti*** quando la funzione termina

QUINDI

- La funzione riceve COPIA dei valori passati dal chiamante
- La funzione ***modifica localmente*** le copie dei valori
- I parametri attuali non saranno mai modificati!

Esempio

Scrivere una funzione che calcoli la potenza N-esima di un valore dato X.

CODIFICA

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

ESEMPIO D'USO

```
main(){
long risultato;
risultato = potenza(4,3);  /* 4^3 = 64 */
}
```

Esempio (cont.)

Il problema precedente può essere risolto anche con la seguente funzione che invece di utilizzare un contatore `i` decrementa il parametro formale `n` ad ogni passo:

```
long potenza(int x, int n) {
    long  p = 1;
    while (n-- > 0)  p *= x;
    return p;
}
```

Stesso **ESEMPIO D'USO**

```
main(){
long risultato;
int b=4;
int e=3;
risultato = potenza(b,e);  /* 4^3 = 64 */
}
```

NOTA

La variabile `e` non viene alterata!

Dichiarazione di funzione in C

Una **dichiarazione di funzione** è costituita dalla *sola intestazione* della funzione, *senza blocco*, terminata da un ; :

<dich-funzione> ::= <intestazione-funzione>;

A cosa serve la dichiarazione?

Per poter usare un servitore!

Infatti il cliente non deve sapere come è fatto deve sapere solo come va chiamato

→ *Non serve* che il cliente conosca *la definizione* della funzione, ma è sufficiente che ne *conosca la dichiarazione*, perché questa contiene *la specifica del protocollo di comunicazione!*

ESEMPIO

- Gli **header** delle librerie (standard) contengono le dichiarazioni di tutte le funzioni incluse nella libreria stessa
- Un programma che voglia usare una determinata funzione/procedura, è sufficiente che includa l'header della libreria che la contiene

Uso delle dichiarazioni nei programmi

Il `main` per poter richiamare una funzione è sufficiente che conosca la sua dichiarazione!

```
#include <stdio.h>
/* contiene le dichiarazioni di printf e scanf */
#include <math.h>
/* contiene la dichiarazione di sqrt */

int max (int, int); /* dichiarazione di max */

int main() {
int a,b,c,d,m;

printf("Inserisci i quattro valori: ");
scanf("%d %d %d %d",&a,&b,&c,&d);
m=max(max(a,b),max(c,d));
printf("Radice di %d: %f",m,sqrt(m));}

/*definizione di max */
int max (int a, int b) { if(a>b) return a; else return b;}
```

Osservazioni su dichiarazione e definizione

- La **dichiarazione** non produce un solo byte di codice quindi, può essere ripetuta più volte (l'importante è che non ci siano due dichiarazioni in contraddizione)
- La **definizione**, invece, contiene il vero codice della funzione non può essere duplicata!! (altrimenti ci sarebbero due codici per la stessa funzione)

QUINDI

Una funzione può essere *dichiarata* più volte, ma *definita* una sola volta.

Esempio

Scrivere una funzione radice che calcoli la radice quadrata (intera) di un valore naturale N.

SPECIFICA

Per ogni numero naturale X, a partire da X=0, calcolare il quadrato X*X fino a quando X*X supera N.
Allora X-1 è la radice quadrata intera di N.

CODIFICA

```
/*restituisce il massimo intero X tale che X*X ≤ N*/
int radice(int n);
int main(){…}
int radice(int n) {
   int x;
   for(x=0; x*x <= n; x++);
   return x-1;
}
```

Esercizio

Scrivere un adeguato main di prova.

Esempio

Scrivere una funzione maiuscolo che, dato un carattere minuscolo, restituisca il corrispondente carattere maiuscolo mentre lascia inalterati gli altri caratteri.

CODIFICA

```
/* restituisce il maiuscolo di c */
char maiuscolo(char c);
int main(){…}
char maiuscolo(char c) {
  if (c<'a' || c>'z') return c;
  return c - 'a' + 'A';
}
```

Esercizio

Scrivere un adeguato main di prova.

Esempio

Scrivere una funzione che calcola il punto medio di due punti in uno spazio bidimensionale.

NOTA

Le strutture vengono passate a funzioni/procedure nel loro insieme, ovvero copiando byte per byte tutto il loro contenuto

CODIFICA

```
#include stdio.h
typedef struct ptx {
         float x, y; } punto;

punto puntoMedio(punto,punto);

void main(){
punto p,q,medio ;

printf(”Inserisci i punti: “);
scanf( "%f %f %f %f", &p.x, &p.y, &q.x,&q.y);
medio=puntoMedio(p,q);}

punto puntoMedio(punto p1, punto p2){
punto pmedio;

pmedio.x=(p1.x+p2.x)/2;
pmedio.y=(p1.y+p2.y)/2 ;
return pmedio;}
```

Esercizio

Scrivere un programma che consenta ad un utente di eseguire operazioni su coppie di punti in uno spazio bidimensionale. Ad ogni iterazione all'utente viene presentato il seguente menu:

0 – calcola il punto medio di una coppia di punti

1 – calcola la lunghezza del segmento delimitato da una coppia di punti

2 – esci

Il programma provvede quindi ad eseguire l'operazione richiesta, chiedendo in input i punti, fino a quando l'utente non digita il valore 2.

Esercizio (cont.)

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

typedef struct{
float x, y;
}punto;

punto acquisisci(void);
punto medio(punto, punto);
float lunghezza(punto, punto);
```

Esercizio (cont.)

```
void main(void) {
int scelta;
punto p1,p2,pm;

do{
printf("Cosa scegli:\n");
printf("0: calcolo del punto medio\n");
printf("1: calcolo della lunghezza del segmento\n");
printf("2: esci\n");
scanf("%d",&scelta);
switch(scelta){
case 0: p1=acquisisci(); p2=acquisisci(); pm=medio(p1,p2);
printf("Punto medio: %f %f\n",pm.x,pm.y);
break;
case 1: p1=acquisisci(); p2=acquisisci();
printf("Lunghezza segmento: %f\n",lunghezza(p1,p2));
case 2:printf("Fine programma!");}}
while(scelta!=2);}
```

Esercizio (cont.)

```
punto acquisisci(){
punto p;

printf("Inserisci il valore in ascissa: ");
scanf("%f",&(p.x));
printf("Inserisci il valore in ordinata: ");
scanf("%f",&(p.y));
return p;}

punto medio(punto a,punto b){
punto c;

c.x=(a.x+b.x)/2;
c.y=(a.y+b.y)/2;
return c;}
```

Esercizio (cont.)

```
float lunghezza(punto a, punto b){
float l1,l2;

l1=fabs(a.x-b.x);
l2=fabs(a.y-b.y);
return(sqrt(l1*l1+l2*l2));}
```

Procedure in C

In C una procedura non è altro che una funzione il cui tipo restituito è `void`

CARATTERISTICHE

- Stessa sintassi di definizione / dichiarazione delle funzioni
- L'istruzione `return` viene usata senza alcuna espressione
  - *deve* solo restituire il *controllo* al chiamante
  - *non deve* restituirgli un *risultato*
- quindi `return` può essere omessa se compare appena prima della parentesi graffa di chiusura blocco

ESEMPI

void stampa_messaggio_errore(int);

Esempio

Si consideri il problema del calcolo della data successiva dove era richiesto il controllo della data. Scrivere una procedura che dato un codice di errore (0: errore nel giorno, 1: errrore nel mese, 2: errore nell'anno), stampi un opportuno messaggio di errore e proporre le modifiche opportune al `main` che risolve il problema.

CODIFICA

```
#include <stdio.h>
void stampaErrore(int);
int main(){
…
if(ANNO>=0) oka = 1; else stampa_errore(2);
if((MESE>=1)&&(MESE<=12)) okm = 1; else stampa_errore(1);
if (GIORNO<=FINEMESE) okg = 1; else stampa_errore(0);
…}
void stampaErrore(int codice){
switch(codice){
        case 0: printf("Giorno inserito errato!"); return;
        case 1: printf("Mese inserito errato!"); return;
        case 2: printf("Anno inserito errato!"); }}
```

Passaggio dei parametri

- Associazione fra parametri attuali e parametri formali che avviene al momento della chiamata di una funzione (o procedura).
- Il meccanismo visto finora si chiama **PASSAGGIO PER VALORE**

I **principali meccanismi** per il passaggio dei parametri sono:

PASSAGGIO PER VALORE-RISULTATO

- I parametri formali contengono una *copia* dei valori dei parametri attuali
- La funzione ***modifica localmente*** le copie dei valori
- I parametri attuali non saranno mai modificati

PASSAGGIO PER INDIRIZZO (O RIFERIMENTO)

- I parametri formali si ***riferiscono*** direttamente alla locazione dei parametri attuali
- Ogni modifica applicata ai parametri formali modifica il valore dei parametri attuali
- Al termine della chiamata il legame viene perso

Passaggio per riferimento in C

Il passaggio per riferimento **NON è disponibile** nella sintassi del linguaggio C ma può essere ***simulato*** tramite **PUNTATORI**

Per realizzarlo, **IL SERVITORE**

- per ogni parametro passato per riferimento, deve specificare che si aspetta il *riferimento* o *puntatore* ad una locazione di memoria.

  La dichiarazione del *parametro formale* ha la seguente forma

  ```
  <nomeTipo> *   <identificatore>
  ```

  <identificatore> è un puntatore ad una locazione di memoria di una variabile di tipo **<nomeTipo>**
- deve accedere esplicitamente all'area di memoria puntata dal parametro formale. La sintassi è la seguente

  ```
  *<identificatore>
  ```

Esempio

Scrivere una procedura che scambi due valori A e B (interi)

SPECIFICA di I° livello

Effettuare una “triangolazione” fra A, B e una variabile ausiliaria.

CODIFICA DEL SERVITORE

```
void scambia(int* PA, int* PB) {
int T;
T = *PA;
*PA = *PB;
*PB = T;}
```

Passaggio per riferimento in C

Per realizzare il passaggio per riferimento in C, **IL CLIENTE**

- deve *ricavare esplicitamente* i riferimenti all'area di memoria dei parametri attuali.

  La sintassi è la seguente

  ```
  &<identificatore>
  ```

  dove **<identificatore>** corrisponde al parametro attuale.

ESEMPIO

Nella libreria `math.h` è contenuta una funzione modf che restituisce la parte intera e la parte frazionaria di un numero double. La sua definizione è:

```
double modf(double x, double *ip);
```

Per usarla in un programma

```
y=modf(x,&z);
```

Esempio

Scrivere una procedura che scambi due valori A e B (interi)

CODIFICA DEL SERVITORE

```
void scambia(int* PA, int* PB) {
int T;
T = *PA;    *PA = *PB;    *PB = T;
  }
```

CODIFICA DEL CLIENTE

```
main(){
int X = 33, Y = 5;
  scambia(&X, &Y);
}
```

Passaggio di vettori

Quando il parametro di una funzione **un vettore**, la lunghezza può essere ommessa perché

- Le funzioni in C non sono in grado di determinare la lunghezza di un vettore passato come argomento

ESEMPIO

Scrivere una funzione che data una stringa, ne restituisce la lunghezza.

```
int lunghezza(char s[]){
   int i;
   for (i=0; s[i]!='\0'; i++) ;
   return i;}
```

- Se la funzione ne ha bisogno, dovremo fornire la lunghezza come argomento

ESEMPIO

Scrivere una funzione dato un vettore di al più 50 elementi di tipo intero, restituisca il valore massimo.

```
int max(int v[],int n){
   int max=v[0];
   for (i=1; i<n; i++)
       if(v[i]>max) max=v[i];
   return max;}
```

Vettori passati come parametri

- I vettori vengono passati ad una funzione/procedura attraverso il loro nome.

ESEMPI

```
main(){
   char s1[] = "Sempre caro mi fu quest'ermo colle";
   int x;
   x = lunghezza(s1);}
main(){
   int vett[5] = {1, 3, 45, 2, 17};
   int m=max(vett,5);}
```

QUINDI

- Quando si passa un vettore a una funzione ***si copia solo il suo indirizzo iniziale*** (`s1=&s1[0]`)!

1. I **vettori** sono quindi sempre **passati per indirizzo**!
2. Una **funzione** non può restituire un vettore!

Esempio

Scrivere la procedura `toUp` che converte una stringa in maiuscolo.

```
void toUp(char s[]){
  int i;
  for (i=0; s[i]!='\0'; i++)
     if(s[i]>='a' && s[i]<='z')
         s[i]=s[i]-'a'+'A'; }
```

Esercizio

Scrivere un adeguato main di prova.