

Tecniche della Programmazione, lez.8

Funzioni e programmazione modulare

What's Modulo?

Sappiamo già che il programma è composto da una collezione di ... parti ...

```
#include <stdio.h>
int main () {
    int n=4;
    ...
    printf ("stampiamo n ... %d\n", n);
    ...
    printf ("\nFINE programma\n");
    return 0;
}
```

- 1) Main program: realizzato da `int main() { ...}`
- 2) Funzioni di libreria: ottenute con le inclusioni di `file header`
 - il programma principale "chiama" una funzione di libreria, "scrivendo" il suo nome e aggiungendo dei *parametri* che verranno usati durante l'esecuzione della funzione
- 3)

What's Modulo?

Sappiamo già che il programma è composto da una collezione di ... parti ...

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

```
int main () {
```

```
    int n=4;
```

```
    ...
```

```
    printf ("stampiamo n ... %d\n", n);
```

```
    ...
```

```
    printf ("\nFINE programma\n");
```

```
    return 0;
```

```
}
```

Per poter usare le funzioni standard di I/O

1) *Main program: realizzato da int main() { ...}*

2) *Funzioni di libreria: ottenute con le inclusioni di file header*

- *il programma principale "chiamà" una funzione di libreria, "scrivendò" il suo nome e aggiungendo dei parametri che verranno usati durante l'esecuzione della funzione*

3)

What's Modulo?

Sappiamo già che il programma è composto da una collezione di ... parti ...

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

```
int main () {
```

```
    int n=4;
```

```
    printf ("stampiamo n ... %d\n", n);
```

```
    ...
```

```
    printf ("\nFINE programma\n");
```

```
    return 0;
```

```
}
```

1) *Main program: realizzato da int*

2) *Funzioni di libreria: ottenute con le inclusioni di file header*

- *il programma principale "chiamà" una funzione di libreria, "scrivendò" il suo nome e aggiungendo dei parametri che verranno usati durante l'esecuzione della funzione*

3)

Chiamata della funzione printf (che, per via dell'include, è "nota al compilatore")

Parametri della chiamata:

- "stampiamo n ... %d\n"
- n

Argomenti è un sinonimo spesso usato per "parametri"

What's Modulo?

Sappiamo già che il programma è composto da una collezione di ... parti ...

```
#include <stdio.h>
int main () {
    int n=4;
    ...
    printf ("stampiamo n ... %d\n", n);
    ...
    printf ("\nFINE programma\n");
    return 0;
}
```

- 1) Main program: realizzato da `int main() { ...}`
- 2) Funzioni di libreria: ottenute con le inclusioni di `file header`
 - il programma principale "chiam" una funzione di libreria, "scrivendo" il suo nome e aggiungendo dei *parametri* che verranno usati durante l'esecuzione della funzione
- 3) Funzioni definite dal programmatore, nel medesimo file della `main()`, o anche in altri file.

What's Modulo? Funzioni definite dal programmatore

```
#include <stdio.h>
int main () {
    int n=4;
    ...
    printf ("stampiamo n ... %d\n", n);
    ...
    printf ("\nFINE programma\n");
    return 0;
}
```

1) Main program: realizzato da `int main() { ...}`

2) Funzioni di libreria: ottenute con le inclusioni di `file header`

- il programma principale "chiamà" una funzione di libreria, "scrivendò" il suo nome e aggiungendo dei *parametri* che verranno usati durante l'esecuzione della funzione

3) **Funzioni definite dal programmatore, nel medesimo file della main(), o anche in altri file.**

3.1) Le **funzioni** sono anche chiamate "**moduli**" del programma:

- corrispondono a "**sottoprogrammi**", ciascuno dei quali risolve, quando viene chiamato ad essere eseguito, un **sottoproblema** del problema generale affrontato dal programma

3.2) Il main program `int main()` è anche lei una funzione

3.3) Tutti i moduli vengono compilati (o sono già compilati, come succede per le funzioni di libreria), e poi, mediante *linking*, si ottiene il programma rilocabile (e poi eseguibile)

What's Modulo? *(sempre funzioni di libreria e def. dal programmatore)*

Le funzioni di libreria risolvono sottoproblemi "generali",

```
printf ("stampiamo la radice di ... %f\n", sqrt(val));
```

```
printf ("n alla m ... %d\n", exp(n,m));
```

```
printf ("valore assoluto di p ... %d\n", fabs(p));
```

Quelle in rosso sono le **chiamate** alle funzioni
in questione

Sottoproblemi particolari

→ **funzioni definite dal programmatore**

Ad esempio

```
maxComunDiv = miaFunzioneMCD(n,m);
```

What's Modulo? ... verbose

Le **funzioni di libreria** risolvono sottoproblemi "generalisti", sottoproblemi che è ben noto possono presentarsi, sono previsti, e quindi sono anche già pre-risolti ...

Ad esempio, le funzioni rese disponibili da `<math.h>` permettono di eseguire calcoli matematici con una semplice chiamata di funzione:

```
printf ("stampiamo la radice di val ... %f\n", sqrt(val));
```

```
printf ("n alla m ... %d\n", exp(n,m));
```

```
printf ("valore assoluto di p ... %d\n", fabs(p));
```

Un problema particolare può presentare invece un sottoproblema non generale, o comunque meno generale della stampa di dati o del calcolo della radice quadrata ... per quel problema non c'è già una soluzione disponibile tramite una funzione di libreria ... allora il programmatore può (o deve) programmare una soluzione al sottoproblema in una **funzione definita dal programmatore** ...

Ad esempio un programma che calcola molti MCD può aver bisogno di una funzione che calcoli l'MCD di due numeri dati: `chiamiamola miaFunzioneMCD`.

Se tale funzione viene progettata ed implementata, allora poi sarà possibile chiamarla in esecuzione ogni volta che serve, così come faremmo con `sqrt()`.

```
maxComunDiv = miaFunzioneMCD(n,m)
```


What's Modulo?

Le funzioni di libreria risolvono sottoproblemi "generali", sottoproblemi che è ben noto possono presentarsi, sono previsti, e quindi sono anche già pre-risolti ...

Ad esempio, le funzioni rese disponibili da `<math.h>` permettono di eseguire calcoli matematici con una semplice chiamata di funzione:

```
printf ("stampiamo la radice di val ... %f\n", sqrt(val));  
    printf ("n alla m ... %d\n", exp(n,m));  
printf ("valore assoluto di p ... %d\n", fabs(p));
```

QUESTE SONO **Chiamate** di funzioni
... di libreria, ma comunque funzioni ...

L'esecuzione della chiamata consiste nella esecuzione della funzione sui dati forniti dai parametri

la chiamata svolge il ruolo di una **ESPRESSIONE** ... il valore risultante viene usato nel contesto in cui la chiamata è stata fatta

Negli esempi sopra le chiamate producono un valore che viene stampato, ma una chiamata può essere anche fatta in altri contesti ...

ad esempio in una istruzione di assegnazione ... `rad = sqrt(val);`
ma può anche direttamente essere una istruzione `stampaMenu();`
`printf("...", ...);`

Altre frecce? Quali altre chiamate ci sono qui? 😊

What's Modulo?

Le funzioni di libreria risolvono sottoproblemi "generali", sottoproblemi che è ben noto possono presentarsi, sono previsti, e quindi sono anche già pre-risolti ...

Ad esempio, le funzioni rese disponibili da `<math.h>` permettono di eseguire calcoli matematici con una semplice chiamata di funzione:

```
printf ("stampiamo la radice di val ... %f\n", sqrt(val));  
printf ("n alla m ... %d\n", exp(n,m));  
printf ("valore assoluto di p ... %d\n", fabs(p));
```

QUESTE SONO **Chiamate** di funzioni

l'esecuzione della chiamata consiste nella esecuzione della funzione sui dati forniti dai parametri

la chiamata svolge il ruolo di una ESPRESSIONE ... il valore risultante viene usato nel contesto in cui la chiamata è stata fatta

In questi esempi le chiamate producono un valore che viene stampato, ma una chiamata può essere anche fatta in altri contesti ...

ad esempio in una istruzione di assegnazione ... `rad = sqrt(val);`

ma può anche direttamente essere una istruzione `stampaMenu();`

`printf("...", ...);`

(in questi ultimi due casi, il valore prodotto dalla chiamata, se c'è, non viene usato dal programma ...)

Funzioni def. dal programmatore

```
#include <stdio.h>
...
definizione miaFunzioneMCD().
...
int main () {
...
    mxComDiv = miaFunzioneMCD(n,m);
...
return 0;
}
```

CHIAMATA

Terminazione della
funzione main()

Con «return» termina
qualsiasi funzione

La funzione `int main()` è quella che usiamo per definire il programma principale cui stiamo lavorando.

Il programma principale è quello che organizza a più alto livello la soluzione del problema. Per risolvere sottoproblemi il programma principale chiama, ove serve, altre funzioni

- ad essere eseguite
- e a produrre così un risultato che è la soluzione del sottoproblema

Quando il programma principale chiama una funzione, questa «prende il controllo dell'esecuzione» ed il programma principale si sospende.

Al termine dell'esecuzione della chiamata, il programma principale riprende la sua esecuzione dal punto in cui si era interrotto.

E questo vale per qualsiasi funzione (ogni funzione potrebbe chiamare in aiuto un'altra funzione, per risolvere un suo sottoproblema ...)

- Main program: realizzato da `int main()`
- Funzioni di libreria: ottenute con le inclusioni
 - il programma principale "chiamerà" una funzione aggiungendo dei parametri che verranno interpretate dal programmatore, ne
- che chiamate "moduli" del programma che risolve, quando viene chiamato ad essere eseguito, un sottoproblema del problema generale affrontato dal programma
- Il main program `int main()` è anche lei una funzione
- Tutti i moduli vengono compilati (o sono già compilati, come succede per le funzioni di libreria), e poi, mediante linking, si ottiene il programma rilocabile (e poi eseguibile)

Vantaggi della programmazione per moduli

- RIUSO:

Avevamo da risolvere il problema **A**; lo abbiamo risolto con un programma in cui abbiamo anche programmato una funzione $f()$ che risolveva il sottoproblema **SP**. Ora stiamo risolvendo il problema **B**, in cui c'è il medesimo sottoproblema **SP**.

Fantastico! (se abbiamo programmato bene $f()$...) possiamo riusare $f()$ anche nel programma che risolve **B**

- TEST

Un programma diviso in moduli può essere testato più efficientemente di un programma che mette in fila tutte le sue istruzioni per pagine e pagine ... (more about that, in future)

- SVILUPPO IN SQUADRA

Se il programma viene progettato in modo da essere scomposto in moduli, ci si può dividere il lavoro con migliori possibilità di successo ...

Funzioni ... $f(x)$, $f(x,y)$, ...

Una **funzione** C è molto simile alla **funzione matematica** che siamo abituati a trattare ... ecco due esempi ... di funzioni matematiche

$$\begin{array}{l} f: \text{Dom} \longrightarrow \text{Codom} \\ a \longmapsto f(a) \end{array}$$

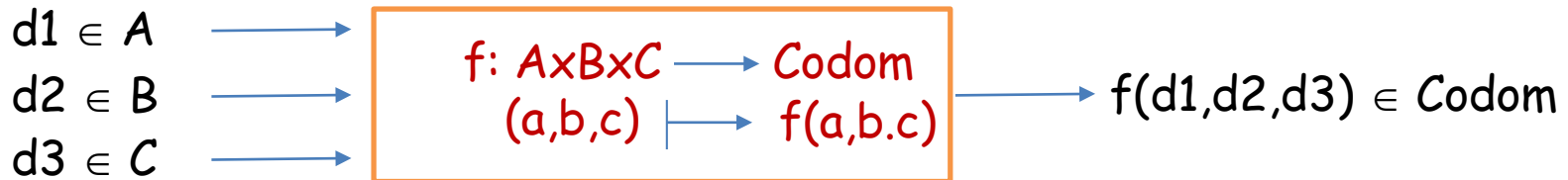
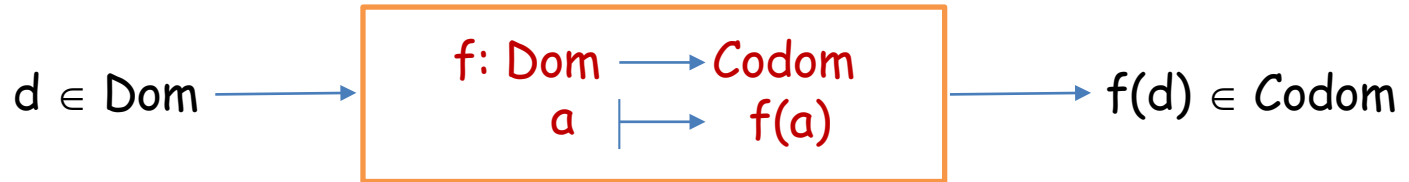
$$\begin{array}{l} f: A \times B \times C \longrightarrow \text{Codom} \\ (a,b,c) \longmapsto f(a,b,c) \end{array}$$

NB

La funzione è come una scatola che fa i calcoli con i parametri che le vengono passati ...

Funzioni ... $f(x)$, $f(x,y)$, ...

Una **funzione** C è molto simile alla **funzione matematica** che siamo abituati a trattare ... ecco due esempi ... di funzioni matematiche



NB

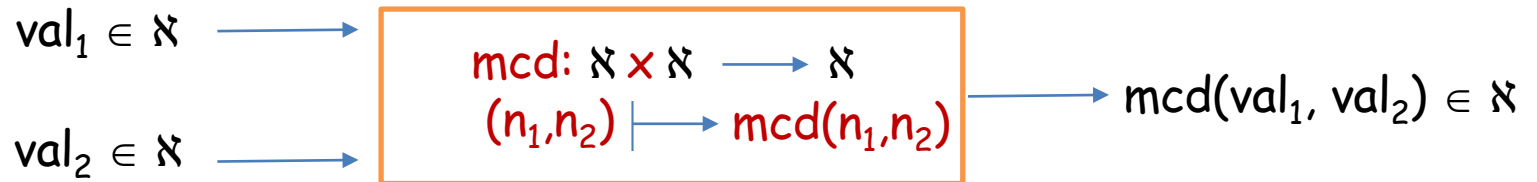
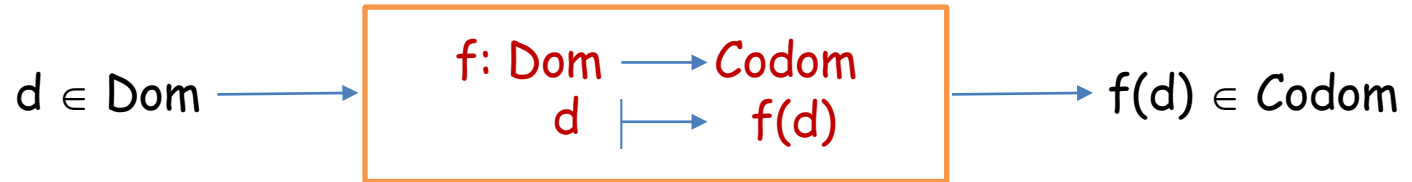
La funzione è come una scatola che fa i calcoli con i parametri che le vengono passati ...

Nella "scatola" a,b,c sono nomi generici per i parametri:

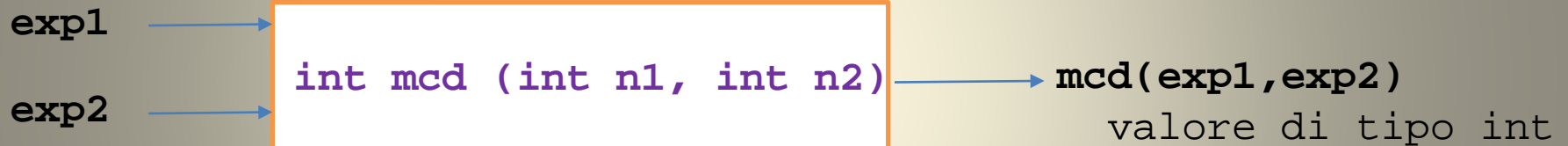
- in generale, la funzione mappa (a,b,c) su $f(a,b,c)$;
- quando eseguiamo il calcolo della funzione con $d1,d2,d3$ viene calcolato il valore di $f(d1,d2,d3)$...

Funzioni ... `funType functionName (paramType_1 param1, ..., paramType_n param_n) {...}`

Una **funzione** C è molto simile alla **funzione matematica** che siamo abituati a trattare ...



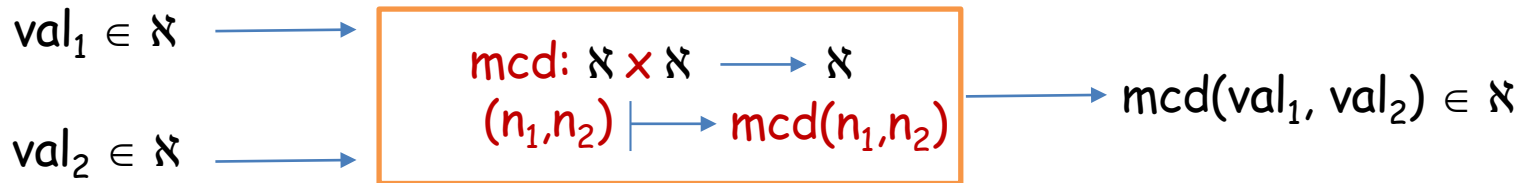
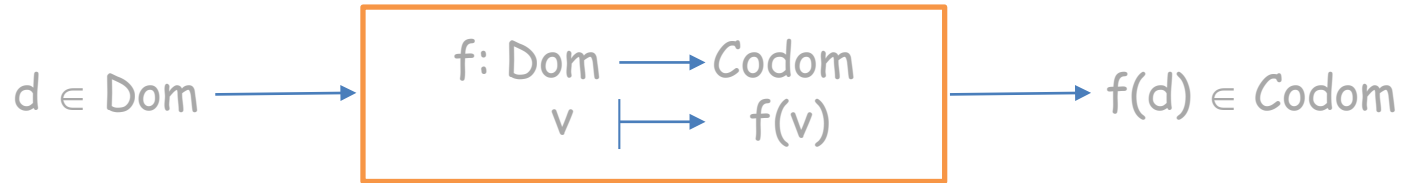
In C



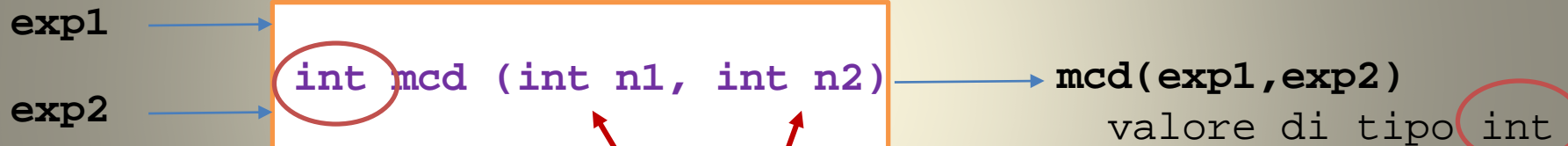
$$\left(\begin{array}{l} \text{mcd}: \text{int} \times \text{int} \longrightarrow \text{int} \\ (n_1, n_2) \longmapsto \text{mcd}(n_1, n_2) \end{array} \right)$$

Funzioni ... `funType functionName (paramType_1 param1, ..., paramType_n param_n) {...}`

Una funzione C è molto simile alla funzione matematica che siamo abituati a trattare ...



In C

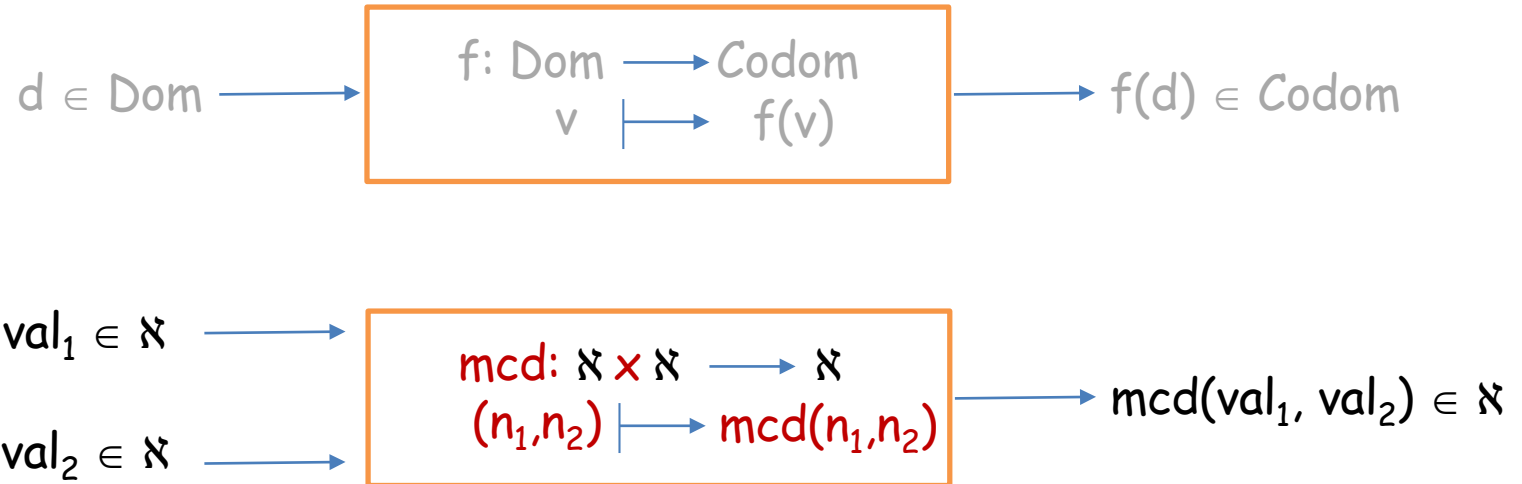


Intestazione della funzione

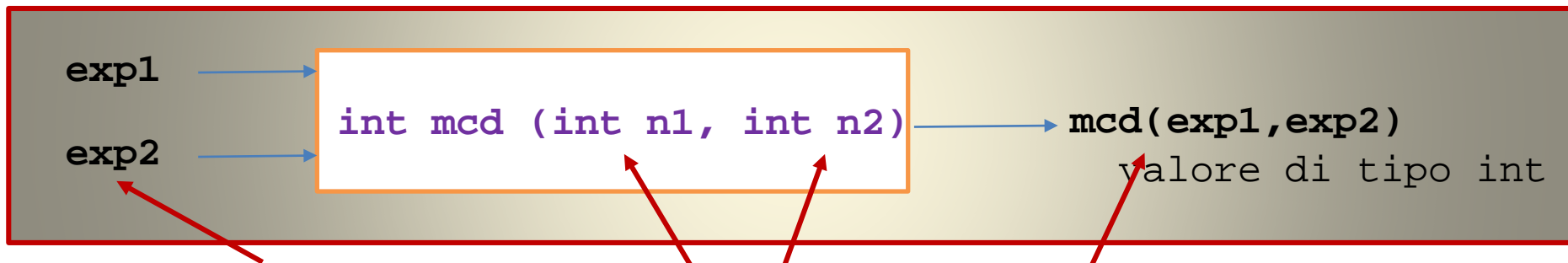
Parametri formali

Funzioni in C

Una funzione C è molto simile alla funzione matematica che siamo abituati a trattare ...



In C



Parametri attuali

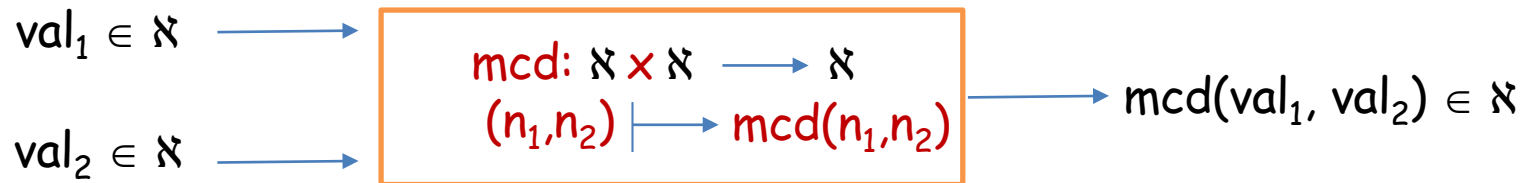
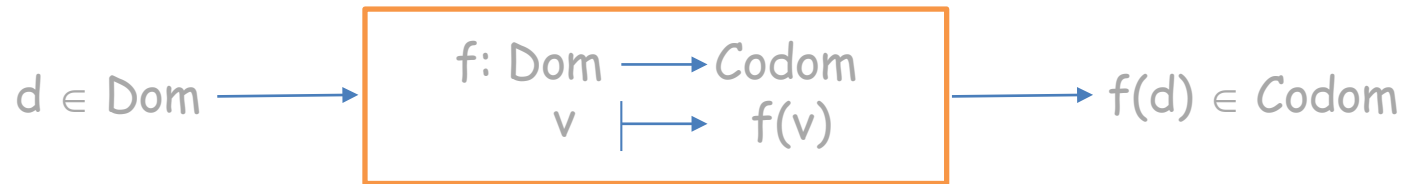
devono essere valori di tipo uguale a quello dei parametri formali

Parametri formali

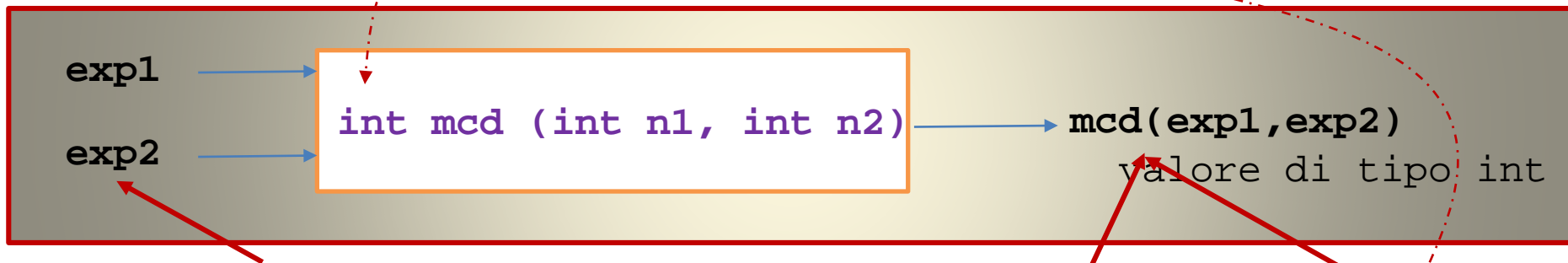
chiamata

Funzioni in C

Una funzione C è molto simile alla funzione matematica che siamo abituati a trattare ...



In C



Parametri attuali

devono essere valori di tipo uguale a quello dei parametri formali

chiamata

Espressione di tipo `int`

Funzioni in C

Una funzione C è molto simile alla funzione matematica che siamo abituati a trattare ...

Quando la FUNZIONE CHIAMANTE effettua la **chiamata** della FUNZIONE CHIAMATA, la prima viene sospesa, la seconda viene eseguita.

Al termine della sua esecuzione, la FUNZIONE CHIAMATA fornisce alla FUNZIONE CHIAMANTE un valore (**VALORE RESTITUITO DALLA CHIAMATA**).

```
mxComDiv = mcd(n,m);
```

Questo valore "appare" nel punto in cui è stata chiamata la funzione.

La chiamata della funzione è in pratica un'espressione, la cui valutazione fornisce il valore ottenuto (RESTITUITO) dall'esecuzione della funzione chiamata.

n

m

```
int mcd (int n1, int n2)
```

`mcd(n,m)` valore
di tipo int

chiamata

Espressione di
tipo int

Parametri attuali

in questo caso due variabili int (che vengono trattate come espressioni e valutate ...)

Funzioni in C

Una funzione C è molto simile alla funzione matematica che siamo abituati a trattare ...

Quando la FUNZIONE CHIAMANTE effettua la chiamata della FUNZIONE CHIAMATA, la prima viene sospesa, la seconda viene eseguita.

Al termine della sua esecuzione, la FUNZIONE CHIAMATA fornisce alla FUNZIONE CHIAMANTE un valore (VALORE RESTITUITO DALLA CHIAMATA).

```
mxComDiv = mcd(n,m);
```

Questo valore "apparirà" nel punto in cui è stata chiamata la funzione.

La chiamata della funzione è in pratica un'espressione, la cui valutazione fornisce il valore ottenuto (RESTITUITO) dall'esecuzione della funzione chiamata.

Il tipo del valore restituito è definito nell'intestazione della funzione.

int n

int m

```
int mcd (int n1, int n2)
```

mcd(n,m) valore
di tipo int

Parametri attuali

Tipo del valore restituito Parametri formali

chiamata

valore restituito

Espressione di tipo int

Un sottoproblema: cubo

Scrivere l'intestazione di una funzione C, `cube()`, che, ricevendo un numero reale, `d`, calcola e restituisce il cubo di `d`.



- Riceve un valore di tipo `double`
- Restituisce un valore di tipo `double`

Un sottoproblema: cubo

Scrivere una funzione che, ricevendo un numero reale, d, calcola e restituisce il cubo di d.

```
double cube (double d)
```

- Riceve un valore di tipo double
- Restituisce un valore di tipo double

Un sottoproblema: cubo

Scrivere una funzione che, ricevendo un numero reale, d , calcola e restituisce il cubo di d .

```
double cube (double d)
```

- Riceve un valore di tipo double
- Restituisce un valore di tipo double

Esempio di programma in cui usarla ...

```
int main () {  
    double num, cub;  
  
    printf("Gentile utente, dammi un ...: ");  
    scanf("%lf", &num);  
  
    cub = cube(num);  
  
    printf("voilà ... %g", cub);  
}
```

Chiamata della funzione `cube`, con parametro attuale `num`;
La chiamata è un'espressione.
In questo caso è una sottoespressione dell'espressione di assegnazione.
Il valore restituito dalla sua valutazione, **di tipo double**, viene assegnato a `cub`.

Un sottoproblema: cubo

Scrivere una funzione che, ricevendo un numero reale, d , calcola e restituisce il cubo di d .

```
double cube (double d)
```

- Riceve un valore di tipo double
- Restituisce un valore di tipo double

Esempio di programma in cui usarla ...

```
int main () {  
    double num, cub;  
  
    printf("Gentile utente, dammi un ...: ");  
    scanf("%lf", &num);  
  
    cub = cube(num);  
  
    printf("voilà ... %g", cub);  
}
```

Chiamata della funzione `cube`, con parametro attuale `num`.
La chiamata è un'espressione.

E anche `num`, il parametro attuale, è un'espressione (la cui valutazione produce semplicemente il valore contenuto in `num`).

E dove sarebbe la funzione cubo?
In quale libreria?

Ah ... La devo fare io?



Sì, ma in questo caso è molto semplice:

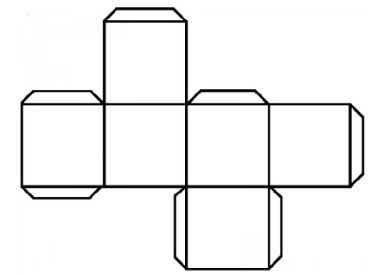
La funzione usa il parametro per calcolare il cubo, e restituisce il valore calcolato al termine.

```
double cube (double d) {  
    double ris;  
  
    ris = d*d*d;  
  
    return(ris);  
}
```

OPPURE, anche

```
double cube (double d) {  
    return(d*d*d);  
}
```

"Programma del cubo", completo



```
/* programma che fa uso della funzione cube */
#include <stdio.h>

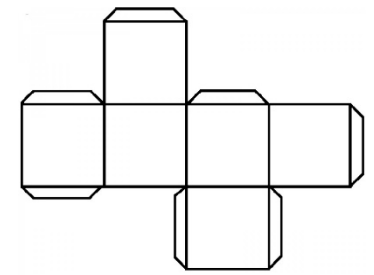
double cube (double d) {
    return(d*d*d);      /* autoesplicativo ... */
}

int main () {
    double num, cub;

    😊

    printf("FINE\n");
    return 0;
}
```

Programma del cubo, completo



```
/* programma che fa uso della funzione cube */
#include <stdio.h>

double cube (double d) {
    return(d*d*d);      /* autoesplicativo ... */
}

int main () {
    double num, cub;

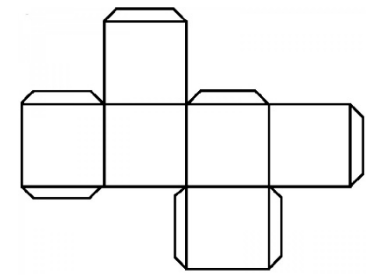
    printf("Gentile utente, ... lo elevo al cubo: ");
    scanf("%lf", &num);

    cub = cube(num);

    printf("voilà ... %g", cub);

    printf("FINE\n");
    return 0;
}
```

Definizione di una funzione



```
/* programma che fa uso della funzione cube */  
#include <stdio.h>
```

```
double cube (double d) {  
    return(d*d*d);      /* autoesplicativo ... */  
}
```

```
int main () {  
    double num, cub;  
  
    printf("Gentile utente, ... lo elevo al cubo: ");  
    scanf("%lf", &num);  
  
    cub = cube(num);  
  
    printf("voilà ... %g", cub);  
  
    printf("FINE\n");  
    return 0;  
}
```

DEFINIZIONE della funzione cube

- Tipo restituito: double
- Tipo del PARAMETRO FORMALE, double
- Nome del PARAMETRO FORMALE, d
- L'esecuzione di return termina la funzione, restituendo alla FUNZIONE CHIAMANTE il valore $d*d*d$

Funzione CHIAMANTE e funzione CHIAMATA

```
/* programma che fa uso della funzione cube */
```

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

```
double cube (double d) {  
    return(d*d*d);      /* autoesplicativo ... */  
}
```

```
int main () {  
    double num, cub;
```

```
printf("Gentile utente, inserisci un numero: ");  
scanf("%lf", &num);
```

```
cub = cube(num);
```

```
printf("voilà ... ");
```

```
printf("FINE\n");  
return 0;
```

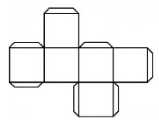
```
}
```

FUNZIONE CHIAMANTE : la main()

CHIAMATA della funzione **cube**

- PARAMETRO ATTUALE **num**
- assegnato al PARAMETRO FORMALE **d**,
- per calcolare **d*d*d**,
cioè
(valore di num)*(valore di num)*(valore di num)
- poi il valore calcolato ...

Funzione CHIAMANTE e funzione CHIAMATA



```
/* programma che fa uso della funzione cube */
#include <stdio.h>

double cube (double d) {
    return(d*d*d);          /* autoesplicativo ... */
}
```

```
int main () {
    double num, cub;
```

```
printf("Gentile ut
scanf("%lf", &num)
```

```
cub = cube(num);
```

```
printf("voilà ...
```

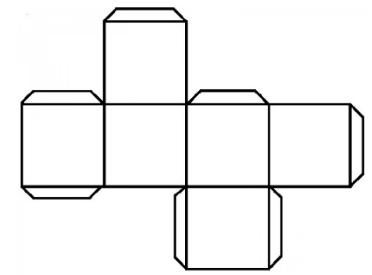
```
printf("FINE\n");
return 0;
}
```

FUNZIONE CHIAMANTE : la main()

CHIAMATA della funzione **cube**

- **PARAMETRO ATTUALE**, il valore dell'espressione **num**
- Il valore del **PARAMETRO ATTUALE** viene usato durante l'esecuzione del codice della funzione;
- questo valore viene assegnato al **PARAMETRO FORMALE** **d**,
- In altre parole questo valore (parametro attuale) viene **USATO COME VALORE DI** **d** (parametro formale) nel codice della funzione.
- quindi quando viene calcolato $d*d*d$, è il cubo del parametro attuale che viene calcolato,
- cioè **(valore di num)*(valore di num)*(valore di num)**
- poi il valore calcolato viene restituito alla funzione chiamante, nel punto della chiamata ... e quindi viene assegnato a **cub**

Programma del cubo, completo



```
/* programma che fa uso della funzione cube */  
#include <stdio.h>
```

```
double cube (double d) {  
    return(d*d*d);      /* autoesplicativo ... */  
}
```

```
int main () {  
    double num;  
  
    printf("Gentile utente, dammi un numero reale  
che te lo elevo al cubo: ");  
    scanf("%lf", &num);  
  
    cub = cube(num);  
  
    printf("voilà ... %g", cub);  
  
    printf("FINE\n");  
    return 0;  
}
```

```
Gentile utente, dammi un numero reale  
che te lo elevo al cubo: 9  
voilà ... 729  
FINE
```

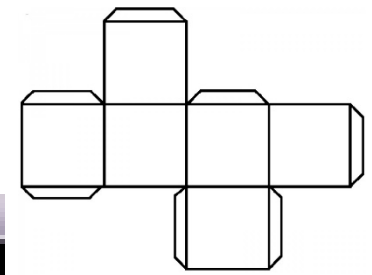
CHIAMATA della funzione cube

- PARAMETRO ATTUALE, il valore dell'espressione **num**
- Il valore del PARAMETRO ATTUALE viene usato durante l'esecuzione del codice della funzione; questo valore viene assegnato al PARAMETRO FORMALE **d**, quindi quando viene calcolato $d*d*d$, è il cubo del parametro attuale che viene calcolato

Sì, l'abbiamo detto meglio nella slide precedente ...

Programma del cubo, completo

```
/* programma che fa uso della funzione cube */
```



```
Gentile utente, dammi un numero reale
che te lo elevo al cubo: 12
voilà' ... 1728
FINE
```

```
*/
```

```
int main () {
    double num,
    printf("Gentile utente, dammi un numero reale
    scanf("%lf", &num);

    cub = cube(num);

    printf("voilà ... %g", cub);

    printf("FINE\n");
    return 0;
}
```


Variabili locali in una funzione

```
double cube (double d) {  
    double ris;          /* variabile locale per il calcolo del cubo */  
  
    ris = d*d*d;  
    return(ris);  
}
```

Il blocco di codice della funzione può contenere dichiarazioni di variabili;

(Tutto come nella main(), che infatti è una funzione ...
La main() è la funzione che viene chiamata ad essere eseguita (aka *invocata*), PER PRIMA, all'atto della messa in esecuzione del programma eseguibile
)

Il codice della funzione `cube`, è costituito da una sequenza di istruzioni strutturate, un blocco ... come per la `main()`.

In queste istruzioni vengono usate le variabili definite nel blocco (si chiamano **VARIABILI LOCALI**) e i **PARAMETRI FORMALI**, per effettuare calcoli e produrre il risultato che la funzione dovrà restituire al momento della terminazione (`return`).

Componenti di una funzione

```
funType functionName (paramType_1 param1, ..., paramType_n param_n) { ... }  
  
double cube (double d) {  
    int ris;  
    ris = d*d*d;  
    return(ris);  
}
```

- **Intestazione:** `double cube (double d)` riassume i tipi e i nomi che dovranno essere usati nella chiamata
- **Blocco:** `{ ... }`: il codice che viene eseguito quando la funzione viene messa in esecuzione; contiene
 - Istruzioni,
 - Dichiarazioni di variabili (**VARIABILI LOCALI**)
 - Una o più istruzioni `return`, che terminano l'esecuzione della funzione, restituendo (se è il caso) un risultato
- **TIPO** del valore **RESTITUITO** come risultato della funzione
- **Nelle funzioni C viene sempre restituito un solo risultato**
- **TIPI** e **NOMI** dei **PARAMETRI FORMALI**: i parametri formali sono usati nella definizione della funzione; sono i simboli usati nel codice e rappresentano (conterranno!) i dati ricevuti dalla funzione al momento della sua messa in esecuzione (cioè i parametri attuali "passati" nella chiamata della funzione)

Chiamata ed Attivazione di funzione - 1/3 -

Un'altra funzione ...

```
int mcd(int n1, int n2) {
    int result;
    while (n1!=n2)
        if (n2>n1)
            n2=n2-n1;
        else n1-=n2;

    result = n1;

return result;
}
```

```
#include <stdio.h>
int main() {
    int primo, secondo, m;

    printf("...numeri... ");
    scanf("%d %d", &primo, &secondo);

    m = mcd(primo, secondo);
    printf("mcd=%d\n", m);
    return 0;
}
```

Chiamata ed Attivazione di funzione - 2/3 -

```
int mcd(int n1, int n2) {  
    int result;  
    while (n1!=n2)  
        if (n2>n1)n2=n2-n;  
        else n1-=n2;  
  
    result = n1;  
  
    return result;  
}
```

funzione mcd,
PARAMETRI FORMALI: n1, n2
VARIABILI LOCALI: result
VALORE RISULTATO di tipo int

```
#include <stdio.h>  
int main() {  
    int primo, secondo, m;  
  
    printf("...numeri... ");  
    scanf("%d %d", &primo, &secondo);  
    m = mcd(primo, secondo);  
    printf("mcd=%d\n", m);  
    return 0;  
}
```

CHIAMATA di funzione,

Chiamata ed Attivazione di funzione - 2/3 -

```
int mcd(int n1, int n2) {  
    int result;  
    while (n1!=n2)  
        if (n2>n1)n2=n2-n;  
        else n1-=n2;  
  
    result = n1;  
  
    return result;  
}
```

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

```
int main() {
```

```
    int primo, secondo, m;
```

```
    printf("...numeri... ");
```

```
    scanf("%d %d", &primo, &secondo);
```

```
    m = mcd(primo, secondo);
```

```
    printf("mcd=%d\n", m);
```

```
    return 0;
```

```
}
```

funzione mcd,

PARAMETRI FORMALI: n1, n2

VARIABILI LOCALI: result

VALORE RISULTATO di tipo int

CHIAMATA di funzione,

- PARAMETRI ATTUALI: primo, secondo

- esecuzione della **FUNZIONE CHIAMANTE:**
SOSPESA

...

Chiamata ed Attivazione di funzione - 2/3 -

```
int mcd(int n1, int n2) {
    int result;
    while (n1!=n2)
        if (n2>n1)n2=n2-n;
        else n1-=n2;

    result = n1;

return result;
}

#include <stdio.h>
int main() {
    int primo, secondo, m;

    printf("...numeri... ");
    scanf("%d %d", &primo, &secondo);
    m = mcd(primo, secondo);
    printf("mcd=%d\n", m);
    return 0;
}
```

funzione mcd,

PARAMETRI FORMALI: n1, n2

VARIABILI LOCALI: result

VALORE RISULTATO di tipo int

CHIAMATA di funzione,

- PARAMETRI ATTUALI: primo, secondo

- esecuzione della **FUNZIONE CHIAMANTE: SOSPESA**

ATTIVAZIONE della **FUNZIONE CHIAMATA**

- passaggio dei parametri:

ATTUALI --> FORMALI

- esecuzione del codice della funzione

- terminazione della funzione chiamata

- "restituzione" del risultato nel punto della chiamata

- ripresa dell'esecuzione della **funzione chiamante**

Chiamata ed Attivazione di funzione - 3/3 -

```
#include <stdio.h>
int main() {
    int primo, secondo;
    ...
    printf("mcd=%d\n", mcd(primo, secondo));
    return 0; }
}
```

CHIAMATA di funzione,

- PARAMETRI ATTUALI: primo, secondo
- esecuzione della FUNZIONE CHIAMANTE: SOSPESA

ATTIVAZIONE della FUNZIONE CHIAMATA

1) Allocazione RDA, in cui parametri FORMALI e variabili locali hanno locazioni dedicate

2)

3)

4)

```
int mcd(int n1, int n2) {
    int result;
    ...
    return result;
}
```

MEMORIA

primo 55

secondo 6

Record di Attivazione:

area di memoria riservata per l'esecuzione di una chiamata

mcd(primo, secondo)



Chiamata ed Attivazione di funzione - 3/3 -

```
#include <stdio.h>
int main() {
    int primo, secondo;
    ...
    printf("mcd=%d\n", mcd(primo, secondo));
    return 0; }
}
```

MEMORIA

primo 5

secondo 6

CHIAMATA di funzione,

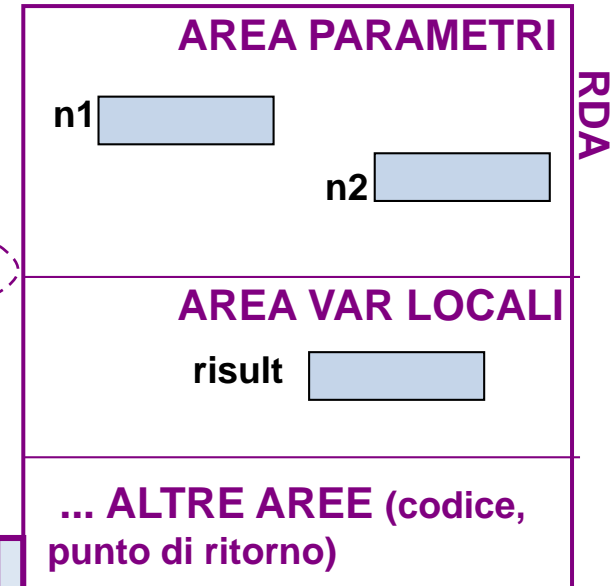
- PARAMETRI ATTUALI: primo, secondo
- esecuzione della FUNZIONE CHIAMANTE: SOSPESA

ATTIVAZIONE della FUNZIONE CHIAMATA

- 1) Allocazione RDA, in cui parametri FORMALI e variabili locali hanno locazioni dedicate
- 2) passaggio dei parametri: i valori dei parametri ATTUALI sono copiati nei parametri FORMALI

Record di Attivazione:
area di memoria riservata per l'esecuzione di una chiamata

mcd(primo, secondo)



RISULTATO

```
int mcd(int n1, int n2) {
    int result;
    ...
    return result; }
}
```


Chiamata ed Attivazione di funzione - 3/3 -

```
#include <stdio.h>
int main() {
    int primo, secondo;
    ...
    printf("mcd=%d\n", mcd(primo, secondo));
    return 0; }
}
```

CHIAMATA di funzione,

- PARAMETRI ATTUALI: primo, secondo
- esecuzione della FUNZIONE CHIAMANTE: SOSPESA

ATTIVAZIONE della FUNZIONE CHIAMATA

- 1) Allocazione RDA, in cui parametri FORMALI e variabili locali hanno locazioni dedicate
- 2) passaggio dei parametri: i valori dei parametri ATTUALI sono copiati nei parametri FORMALI

```
int mcd(int n1, int n2) {
    int result;
    ...
    return result;
}
```

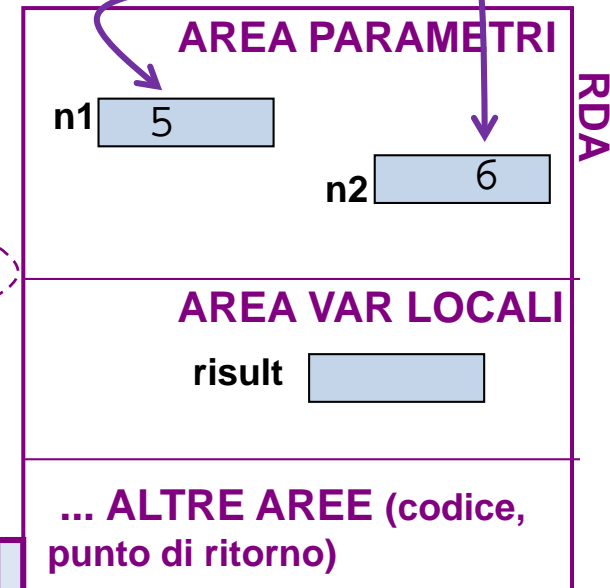
MEMORIA

primo 5

secondo 6

Record di Attivazione:
area di memoria riservata per l'esecuzione di una chiamata

mcd(primo, secondo)



RISULTATO

Chiamata ed Attivazione di funzione - 3/3 -

```
#include <stdio.h>
int main() {
    int primo, secondo;
    ...
    printf("mcd=%d\n", mcd(primo, secondo));
    return 0; }
}
```

MEMORIA

primo 5

secondo 6

CHIAMATA di funzione,

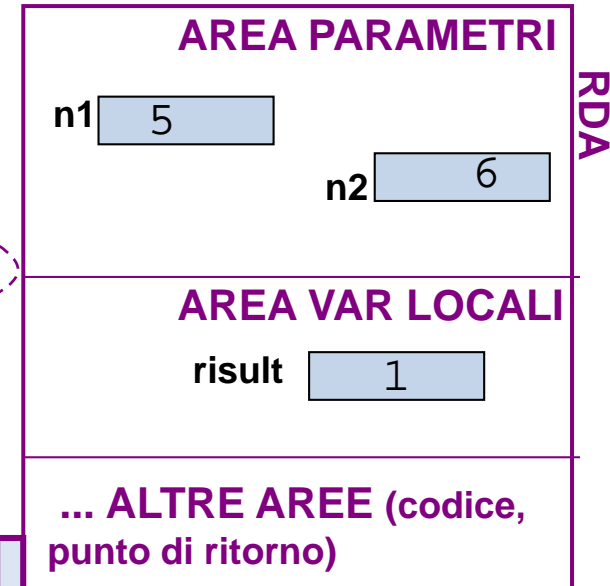
- PARAMETRI ATTUALI: primo, secondo
- esecuzione della FUNZIONE CHIAMANTE: SOSPESA

ATTIVAZIONE della FUNZIONE CHIAMATA

- 1) Allocazione RDA, in cui parametri FORMALI e variabili locali hanno locazioni dedicate
- 2) passaggio dei parametri: i valori dei parametri ATTUALI sono copiati nei parametri FORMALI
- 3) esecuzione del codice della funzione nel RDA (usa solo parametri formali e variabili locali)
- 4)

Record di Attivazione:
area di memoria riservata per l'esecuzione di una chiamata

mcd(primo, secondo)



RISULTATO

```
int mcd(int n1, int n2) {
    int result;
    ...
    return result;
}
```

Chiamata ed Attivazione di funzione - 3/3 -

```
#include <stdio.h>
int main() {
    int primo, secondo;
    ...
    printf("mcd=%d\n", mcd(primo, secondo));
    return 0; }
}
```

MEMORIA

primo 5

secondo 6

CHIAMATA di funzione,

- PARAMETRI ATTUALI: primo, secondo
- esecuzione della FUNZIONE CHIAMANTE: SOSPESA

ATTIVAZIONE della FUNZIONE CHIAMATA

- 1) Allocazione RDA, in cui parametri FORMALI e variabili locali hanno locazioni dedicate
- 2) passaggio dei parametri: i valori dei parametri ATTUALI sono copiati nei parametri FORMALI
- 3) esecuzione del codice della funzione nel RDA (usa solo parametri formali e variabili locali)
- 4) terminazione della funzione chiamata
 - restituzione del risultato
 - deallocazione RDA
 - ripresa della funzione chiamante

Record di Attivazione:

area di memoria riservata per l'esecuzione di una chiamata

mcd(primo, secondo)

AREA PARAMETRI

n1 5

n2 6

RDA

AREA VAR LOCALI

result 1

... ALTRE AREE (codice, punto di ritorno)

RISULTATO

```
int mcd(int n1, int n2) {
    int result;
    ...
    return result;
}
```

Chiamata ed Attivazione di funzione - 3/3 -

```
#include <stdio.h>
int main() {
    int primo, secondo;
    ...
    printf("mcd=%d\n", mcd(primo, secondo));
    return 0; }
}
```

MEMORIA

primo 5

secondo 6

CHIAMATA di funzione,

- PARAMETRI ATTUALI: primo, secondo
- esecuzione della FUNZIONE CHIAMANTE: SOSPESA

ATTIVAZIONE della FUNZIONE CHIAMATA

- 1) Allocazione RDA, in cui parametri FORMALI e variabili locali hanno locazioni dedicate
- 2) passaggio dei parametri: i valori dei parametri ATTUALI sono copiati nei parametri FORMALI
- 3) esecuzione del codice della funzione nel RDA (usa solo parametri formali e variabili locali)
- 4) terminazione della funzione chiamata
 - restituzione del risultato
 - deallocazione RDA
 - ripresa della funzione chiamante

RDA deallocato ...

Non c'è più

Non ci sono più le locazioni dedicate a n1, n2, result ...

```
int mcd(int n1, int n2) {
    int result;
    ...
    return result;
}
```

controlli sulle chiamate di funzione

Il compilatore determina se una chiamata è legittima, in base

- al fatto di conoscere la funzione (aver visto definizione)
- e ai tipi di dati coinvolti nell'istruzione in cui appare la chiamata

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

```
int mcdFun (int n1, int n2) {  
    while (n1 != n2)  
        if (n2>n1)  
            n2=n2-n1;  
        else n1-=n2;  
return(n1);  
}
```

```
int main () {                                int n, m, mcd;  
    printf("fornire i due numeri: ");  
    scanf("%d %d", &n, &m);  
  
    mcd = mcdFun(n,m);  
    printf("il massimo comun divisore tra %d e %d", n,m);  
    printf(" è %d\n", mcd);                /* prosecuzione output */  
...  
}
```

controlli sulle chiamate di funzione

Il compilatore determina se una chiamata è legittima, in base

- al fatto di conoscere la funzione
- e ai tipi di dati coinvolti nell'istruzione in cui appare la chiamata

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

```
int mcdFun (int n1, int n2)
```

```
while (n1 < n2)
```

```
if (n2 > n1)
```

```
n2 = n1;
```

```
else n1 = n2;
```

```
return(n1);
```

```
}
```

```
int main () {
```

```
printf("form");
```

```
scanf("%d %d", &n, &m);
```

```
mcd = mcdFun(n,m);
```

```
printf("il massimo comun divisore tra %d e %d", n,m);
```

```
printf(" è %d\n", mcd); /* prosecuzione output */
```

...

CHIAMATA della funzione mcd, che avviene nell'ambito di una istruzione di assegnazione presente nel programma;

Quando il programma viene compilato, la legittimità delle istruzioni viene controllata; in particolare viene controllata la legittimità di questa chiamata:

a "tempo di compilazione" il Compilatore

- conosce la funzione mcdFun: sa che si chiama mcdFun, che restituisce un valore intero e che riceve due valori interi (quando il compilatore arriva qui ha già visto la definizione della funzione ... ci è già "passato sopra" ... infatti la definizione sta più su di qui nel file ...)
- controlla la chiamata e decide che
 - la chiamata di mcdFun(n,m) è corretta (ha parametri attuali del numero e tipo giusto)
 - il risultato della chiamata è del tipo giusto per l'uso che se ne vuol fare nell'istruzione di assegnazione

Quindi la chiamata è legittima

controlli sulle chiamate di funzione - 2

Il compilatore determina se una chiamata è legittima, in base

- al fatto di conoscere la funzione
- e ai tipi di dati coinvolti nell'istruzione in cui appare la chiamata

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

```
int mcdFun (int n1, int n2) {  
    while (n1 != n2)  
        if (n2>n1)  
            n2=n2-n1;  
        else n1-=n2;  
return(n1);  
}
```

```
int main () {          int n, m, mcd, p;  
    printf("fornire i due numeri: ");  
    scanf("%d %d", &n, &m);
```

```
... mcdFun(n);          /* 😊 */  
... mcdFn(n,m);        /* 😊 */  
... mcdFun(n, m, p);   /* 😊 */
```

controlli sulle chiamate di funzione - 2

Il compilatore determina se una chiamata è legittima, in base

- al fatto di conoscere la funzione
- e ai tipi di dati coinvolti nell'istruzione in cui appare la chiamata

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

```
int mcdFun (int n1, int n2) {  
    while (n1 != n2)  
        if (n2>n1)  
            n2=n2-n1;  
        else n1-=n2;  
return(n1);  
}
```

```
int main () {          int n, m, mcd, p;  
    printf("fornire i due numeri: ");  
    scanf("%d %d", &n, &m);
```

```
... mcdFun(n);          /* non corretta (too few parameters) */  
... mcdFn(n,m);        /* linker error (funzione non nota) */  
... mcdFun(n, m, p);   /* non corretta (too many ...)*/
```


controlli sulle chiamate di funzione - 3

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

```
int main () {  
    double num, cub;  
    printf("Gentile utente, dammi un numero ...: ");  
    scanf("%lf", &num);  
  
    cub = cube(num);  
  
    printf("voilà ... %g\n", cub);  
  
    printf("FINE\n");  
    return 0;  
}  
  
double cube (double d) {  
    double ris;  
  
    ris = d*d*d;  
    return(ris);  
}
```

PROVARE QUESTO
PROGRAMMA ... CHE
ERRORI DA`???

controlli sulle chiamate di funzione - 3

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

```
int main () {           double num, cub;  
    printf("Gentile utente, dammi un numero ...: ");  
    scanf("%lf", &num);
```

Qui c'è l'inizio del problema:
cube non è nota al compilatore al momento della chiamata;
è un errore cui il compilatore tenta di porre rimedio: i tipi coinvolti
vengono assunti (per convenzione) tutti come int.

```
    cub = cube(num);
```

```
    printf("voilà ... %g\n", cub);
```

```
    printf("FINE\n");  
    return 0;  
}
```

Qui c'è l'altro estremo del problema: per quel che sa il compilatore la
funzione è `int cube(int par)`
quindi questa definizione è sorprendente e provoca un errore di
compilazione

```
double cube (double d) {  
    double ris;
```

```
    ris = d*d*d;  
    return(ris);  
}
```

MORALE: le funzioni vanno definite, o *dichiarate*, prima che qualche altra
funzione le chiami.
Se non vogliamo definire la funzione in cima al file, possiamo limitarci a
dichiararla ...

definizione Vs. DICHIARAZIONE di funzione

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

```
double cube (double );
```

```
int main () {           double num, cub;
    printf("Gentile utente, dammi un numero ...: ");
    scanf("%lf", &num);

    cub = cube(num);
    printf("voilà ... %g\n", cub);

    printf("FINE\n");
    return 0;
}
```

```
/* DEFINIZIONE di cube()*/
```

```
double cube (double d) {
    double ris;
    ris = d*d*d;
    return(ris);
}
```

Questa è una **dichiarazione** della funzione cube;
dichiara, per il compilatore, qual è la intestazione della
funzione, così i tipi del risultato e dei parametri sono noti al
compilatore, prima che la funzione cube venga usata/chiamata

nella dichiarazione servono

- nome della funzione
- tipo del risultato
- tipi dei parametri

i nomi dei parametri non sono necessari nella dichiarazione (ma si possono mettere se si vuole, tanto il compilatore non sa che farsene)

```
double cube (double d)
```

```
double cube (double pippo)
```

(I nomi dei parametri sono evidentemente necessari nella
DEFINIZIONE)

Tipo void

void è il "tipo nullo".

è utile soprattutto (almeno per ora) quando si vogliono definire funzioni che non devono restituire un risultato alla funzione chiamante.

```
/* funzione che si occupa di stampare sul video molti 47 */
```

```
void stampaMolti47 (int q) {  
    int i;
```

```
    for (i=0; i<q; i++)  
        printf("sempre lui: %d\n", fun47());
```

```
return; /* nulla da restituire come risultato: return per  
terminare l'esecuzione della funzione e basta */  
}
```

```
int fun47 () {  
return 47;  
}
```

```
int menoFun47 (void) {  
return 47;  
}
```

void si può usare anche per specificare che "non ci sono parametri formali in una funzione" (ma per questo basta non mettere nulla tra le parentesi ...)

Tipo void - 2

Programma che usa le funzioni definite prima

```
void menu();
int fun47 ();
int menoFun47 (void);
void stampaMolti47 (int );

int main () {
    int scelta, quanteVolte;

    printf(" Stimato/a utente, ...: \n");

    menu();
    scanf("%d", &scelta);

    if (scelta==1)
...

```

```
C:\Users\marco\Desktop\lezione10\voids.exe
Stimato/a utente, esegui una scelta scrivendo un numero intero
secondo quanto di seguito indicato:
--- Scrivi 1 per vedere il numero
--- Scrivi 2 per vedere il numero con una soluzione meno bella
--- Scrivi 3 per vedere il numero tante volte
    (Hai solo queste scelte ...)
1
sempre lui: 47
FINE
```

Tipo void - 3

```
/* funzione per stampare un menù di scelte */  
void menu() {  
    printf(" --- Scrivi 1 per vedere il numero\n");  
    printf(" --- Scrivi 2 per vedere il numero con una soluzione meno bella  
                                                \n");  
    printf(" --- Scrivi 3 per vedere il numero tante volte\n");  
    printf(" (Hai solo queste scelte ...)\n");  
return; /* nulla da restituire come risultato: return per  
        terminare l'esecuzione della funzione e basta */  
}
```

```
C:\Users\marco\Desktop\lezione 8\void3.exe  
Stimato/a utente, esegui una scelta scrivendo un numero intero  
secondo quanto di seguito indicato:  
--- Scrivi 1 per vedere il numero  
--- Scrivi 2 per vedere il numero con una soluzione meno bella  
--- Scrivi 3 per vedere il numero tante volte  
    (Hai solo queste scelte ...)  
1  
sempre lui: 47  
FINE
```

Tipo void - 4

```
void menu();
int fun47 ();
int menoFun47 (void)
void stampaMolti47 (
int main () {
    int scelta, quanteVolte;
    printf(" Stimato/a utente, esegui una scelta scrivendo un numero intero
secondo quanto di seguito indicato:
--- Scrivi 1 per vedere il numero
--- Scrivi 2 per vedere il numero con una soluzione meno bella
--- Scrivi 3 per vedere il numero tante volte
(Hai solo queste scelte ...)
1
sempre lui: 47
FINE

    menu();
    scanf("%d", &scelta);

    if (scelta==1)
        printf("sempre lui: %d\n", fun47());
    else if (scelta==2)
        printf("ancora e sempre lui: %d\n", menoFun47());
    else if (scelta==3) {
        printf("ah! Dimmi quante volte ...: ");
        scanf("%d", &quanteVolte);
        stampaMolti47(quanteVolte);
    }
    else ... ?
```

chiamate

Tipo void - 5

```
void menu();
int fun47 ();
int menoFun47 (void)
void stampaMolti47 (
int main () {
    int scelta, quante
    printf(" Stimato/a
```

```
Stimato/a utente, esegui una scelta scrivendo un numero intero
secondo quanto di seguito indicato:
--- Scrivi 1 per vedere il numero
--- Scrivi 2 per vedere il numero con una soluzione meno bella
--- Scrivi 3 per vedere il numero tante volte
(Hai solo queste scelte ...)
2
ancora e sempre lui: 47
FINE
```

```
menu();
scanf("%d", &scelta);
```

```
if (scelta==1)
    printf("sempre lui: %d\n", fun47());
else if (scelta==2)
    printf("ancora e sempre lui: %d\n", menoFun47());
else if (scelta==3) {
    printf("ah! Dimmi quante volte ...: ");
    scanf("%d", &quanteVolte);
    stampaMolti47(quanteVolte);
}
else ... ?
```


Tipo void - 6

```
void menu();
int fun47 ();
int menoFun47 (
void stampaMolt

int main () {
    int scelta, q
    printf(" Stim
    menu();
    scanf("%d", &
    if (scelta==
        printf("s
    else if (scelta==2)
        printf("ancora e sempre lui: %d\n", menoFun47());
    else if (scelta==3) {
        printf("ah! Dimmi quante volte ...: ");
        scanf("%d", &quanteVolte);
        stampaMolti47(quanteVolte);
    }
    else ... ?
```

Stimato/a utente, esegui una scelta scrivendo un numero intero secondo quanto di seguito indicato:

- Scrivi 1 per vedere il numero
- Scrivi 2 per vedere il numero con una soluzione meno bella
- Scrivi 3 per vedere il numero tante volte (Hai solo queste scelte ...)

ah! Dimmi quante volte vorresti veder stampato quel numero: 7

sempre lui: 47

sempre lui: 47

sempre lui: 47

sempre lui: 47

sempre lui: 47

sempre lui: 47

sempre lui: 47

sempre lui: 47

FINE

Tipo void - 6

```
void menu();
int fun47 ();
int menoFun47
void stampaMo
```

```
int main () {
    int scelta,
    printf(" St
```

```
Stimato/a utente, esegui una scelta scrivendo un numero intero
secondo quanto di seguito indicato:
--- Scrivi 1 per vedere il numero
--- Scrivi 2 per vedere il numero con una soluzione meno bella
--- Scrivi 3 per vedere il numero tante volte
(Hai solo queste scelte ...)
42
come hai potuto? Scelta sbagliata. Addio.
FINE
```

```
menu();
```

```
scanf("%d", &scelta);
```

```
if (scelta==1)
```

```
    printf("sempre lui: %d\n", fun47());
```

```
else if (scelta==2)
```

```
    printf("ancora e sempre lui: %d\n", menoFun47());
```

```
else if (scelta==3) {
```

```
    printf("ah! Dimmi quante volte ...: ");
```

```
    scanf("%d", &quanteVolte);
```

```
    stampaMolti47(quanteVolte);
```

```
}
```

```
else ... ? ☺
```

Tipo void - 7

```
void menu();  
int fun47 ();  
int menoFun47 (  
void stampaMolt
```

```
int main () {  
    int scelta, q  
    printf(" Stim
```

```
    menu();  
    scanf("%d", &
```

```
Stimato/a utente, esegui una scelta scrivendo un numero intero  
secondo quanto di seguito indicato:  
--- Scrivi 1 per vedere il numero  
--- Scrivi 2 per vedere il numero con una soluzione meno bella  
--- Scrivi 3 per vedere il numero tante volte  
(Hai solo queste scelte ...)  
42  
come hai potuto? Scelta sbagliata. Addio.  
FINE
```

```
    if (scelta==1)  
        printf("sempre lui: %d\n", fun47());  
    else if (scelta==2)  
        printf("ancora e sempre lui: %d\n", menoFun47());  
    else if (scelta==3) {  
        printf("ah! Dimmi quante volte ...: ");  
        scanf("%d", &quanteVolte);  
        stampaMolti47(quanteVolte);  
    }  
    else printf("come hai potuto? Scelta sbagliata. Addio.\n");  
  
    printf("FINE\n");  
    return 0;  
}
```

potenze - 0

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

```
double potenza (double, int );    /* dichiarazione della funzione che calcola
```

```
caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 3 4
--- 3 elevato alla 4 = 81

caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 4 3
--- 4 elevato alla 3 = 64

caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 12321 1000
... ok, torna quando vuoi!

FINE programma
print ( "FINE programma" );
return 0;
}
```

```
double potenza(double num, int exp) {
...
return ris;
}
```

potenze - 0

Algoritmo: leggiamo numero ed esponente tante volte; ogni volta calcoliamo e stampiamo la potenza di numero elevato ad esponente; se l'esponente è mille non si ripete più

0) numero, esponente; potenza;

1) ripeti

1.1) chiedere/leggere numero ed esponente

1.2) se esponente non è 1000

1.2.1)

1.2.2)

1.2.3)

altrimenti

1.2.4) stampare un addio

mentre esponente è diverso da 1000

2) fine programma

potenze - 0

Algoritmo: leggiamo numero ed esponente tante volte; ogni volta calcoliamo e stampiamo la potenza di numero elevato ad esponente; se l'esponente è mille non si ripete più

0) numero, esponente; potenza;

1) ripeti

1.1) chiedere/leggere numero ed esponente

1.2) se esponente non è 1000

1.2.1) calcolare numero elevato ad esponente

1.2.2) e metterlo in una variabile potenza

1.2.3) e stampare potenza

altrimenti

1.2.4) stampare un addio

mentre esponente è diverso da 1000

2) fine programma

potenze - 0

Algoritmo: leggiamo numero ed esponente tante volte; ogni volta calcoliamo e stampiamo la potenza di numero elevato ad esponente; se l'esponente è mille non si ripete più

0) numero, esponente; potenza;

1) ripeti

1.1) chiedere/leggere numero ed esponente

1.2) se esponente non è 1000

1.2.1) calcolare numero elevato ad esponente

1.2.2) e metterlo in una variabile potenza

1.2.3) e stampare potenza

altrimenti

1.2.4) stampare un addio

mentre esponente è diverso da 1000

2) fine programma

Però, "calcolare numero elevato ad esponente è un problema a sé stante ... un sottoproblema che vale la pena di studiare a parte

Algoritmo per calcoloPotenza(num, esp)

0) serve una var ris; si fa ... ris * num ... circa esp volte ...

1) ... dopo 😊

2) ...

3)

potenze - 0

```
#include <stdio.h>

double potenza (double, int );    /* dichiarazione della funzione che calcola
                                   la potenza di un numero elevato ad un esponente */

int main () {
    int esponente;
    double numero, potenza;

    do {
        printf ("caro/a utente, ... 1000 per finire): ");
        scanf("%lf %d", &numero, &esponente);

        if (esponente!=1000) {
            potenza = potenza(numero, esponente);
            printf (" --- %g elevato alla %d = %g \n\n", numero, esponente, potenza);
        }
        else printf ("... ok, torna quando vuoi!\n\n");
    } while (esponente!=1000);

    printf ("\nFINE programma\n");
    return 0;
}

double potenza(double num, int exp) {
    ...
    return ris;
}
```


potenze - 1

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

```
double potenza (double, int );    /* dichiarazione della funzione che calcola  
la potenza di un numero elevato ad un esponente */
```

```
int main () {  
    int esponente;  
    double numero, potenza;  
  
    do {  
        printf ("caro/a utente, ... 1000 per finire): ");  
        scanf("%lf %d", &numero, &esponente);  
  
        if (esponente!=1000) {  
            potenza = potenza(numero, esponente);  
            printf (" --- %g elevato alla %d = %g \n\n", numero, esponente, potenza);  
        }  
        else printf ("... ok, torna quando vuoi!\n\n");  
    } while (esponente!=1000);
```

**whoops, potenza variabile e
potenza funzione
confondono il compilatore ...**

```
printf ("\nFINE programma\n");  
return 0;  
}
```

```
double potenza(double num, int exp) {  
...  
return ris;  
}
```

potenze - 2

```
#include <stdio.h>

double potenza (double, int );    /* dichiarazione della funzione che calcola
                                   la potenza di un numero elevato ad un esponente */

int main () {
    int esponente;
    double numero, potenzaVal;

    do {
        printf ("caro/a utente, ... 1000 per finire): ");
        scanf("%lf %d", &numero, &esponente);

        if (esponente!=1000) {
            potenzaVal = potenza(numero, esponente);
            printf (" --- %g elevato alla %d = %g \n\n", numero, esponente, potenzaVal);
        }
        else printf ("... ok, torna quando vuoi!\n\n");
    } while (esponente!=1000);

    printf ("\nFINE programma\n");
    return 0;
}

double potenza(double num, int exp) {
    ...
    return ris;
}
```

potenze - 3 - alternativa praticabile (ma non seguita ora)

```
#include <stdio.h>

double potenza (double, int );    /* dichiarazione della funzione che calcola
                                   la potenza di un numero elevato ad un esponente */

int main () {
    int esponente;
    double numero, potenzaVal;

    do {
        printf ("caro/a utente, ... 1000 per finire): ");
        scanf("%lf %d", &numero, &esponente);

        if (esponente!=1000) {
            potenzaVal = potenza(numero, esponente);
            printf (" --- %g elevato alla %d = %g \n\n", numero, esponente,
                potenzaVal potenza(numero, esponente));
        }
        else printf ("... ok, torna quando vuoi!\n\n");
    } while (esponente!=1000);

    printf ("\nFINE programma\n");
    return 0;
}

...
```

potenze - 4

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

```
double potenza (double
```

```
int main () {
```

```
    int esponente;
```

```
    double numero, pot
```

```
do {
```

```
    printf ("caro/a u
```

```
    scanf ("%lf %d", &
```

```
    if (esponente!=1000) {
```

```
        potenz
```

```
        print:
```

```
    }
```

```
    else pri
```

```
    } while (es
```

```
printf ("\n")
```

```
return 0;
```

```
}
```

```
double potenza(double num, int exp) {
```

```
...
```

```
return ris;
```

```
}
```

```
caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 3 4
--- 3 elevato alla 4 = 81
```

```
caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 4 3
--- 4 elevato alla 3 = 64
```

```
caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 12321 1000
... ok, torna quando vuoi!
```

```
FINE programma
```

Algoritmo per calcolare la potenza di num elevato a exp

0) serve la variabile ris

1) 😊

2) 😊

3) restituire ris alla funzione chiamante

potenze - 4

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

```
double potenza (double
```

```
int main () {
```

```
    int esponente;
```

```
    double numero, pot
```

```
do {
```

```
    printf ("caro/a u
```

```
    scanf("%lf %d", &
```

```
if (esponente!=1000) {
```

```
    potenz
```

```
    print:
```

```
}
```

```
else pri
```

```
} while (es
```

```
printf ("\n
```

```
return 0;
```

```
}
```

```
double potenza(double num, int exp) {
```

```
...
```

```
return ris;
```

```
}
```

```
caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 3 4
--- 3 elevato alla 4 = 81
```

```
caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 4 3
--- 4 elevato alla 3 = 64
```

```
caro/a utente, dammi un reale e un esponente intero
che ti calcolo la potenza (esponente=1000 per finire): 12321 1000
... ok, torna quando vuoi!
```

```
FINE programma
```

Algoritmo per calcolare la potenza di num elevato a exp

0) serve la variabile ris

1) inizializzare ris = num;

2) per exp-1 volte, moltiplicare ris per num

3) restituire ris alla funzione chiamante



potenze - 5

```
#include <stdio.h>

double potenza (double, int ); /* dichiarazione della funzione che calcola
la potenza di un numero elevato ad un esponente */

int main () {
    int esponente;
    double numero, potenzaVa

do {
    printf ("caro/a utente,
scanf("%lf %d", &numero

    if (esponente!=1000) {
        potenza = potenza(num
        printf (" --- %g ele
    }
    else printf ("... ok, torna quando vuoi!\n\n");
} while (esponente!=1000);

printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo per calcolare la potenza di num elevato a exp

0) serve la variabile ris

1) inizializzare ris = num;

2) per exp-1 volte, moltiplicare ris per num e assegnare a ris il risultato

3) restituire ris alla funzione chiamante

```
double potenza(double num, int exp) {
    double ris=num;
    int i;

    for (i=1; i<exp; i++) {
        ris *= num; /* ris = ris * num; */
    }
    return ris;
}
```

potenze - 5 - alternativa interessante

```
#include <stdio.h>

double potenza (double, int ); /* dichiarazione della funzione che calcola
la potenza di un numero elevato ad un esponente */

int main () {
    int esponente;
    double numero, potenzaVa

do {
    printf ("caro/a utente,
scanf("%lf %d", &numero

    if (esponente!=1000) {
        potenza = potenza(num
        printf (" --- %g ele
    }
    else printf ("... ok, torna quando vuoi!\n\n");
} while (esponente!=1000);

printf ("\nFINE programma\n");
return 0;
}
```

Algoritmo per calcolare la potenza di num elevato a exp

0) serve la variabile ris

1) inizializzare ris = 1;

2) per exp volte, moltiplicare ris per num e assegnare a ris il risultato

3) restituire ris alla funzione chiamante

```
double potenza(double num, int exp) {
    double ris=1;
    int i;

    for (i=0; i<exp; i++) {
        ris *= num; /* ris = ris * num; */
    }
    return ris;
}
```

Visibilità delle variabili e dei parametri

Chi può usare le variabili locali di una funzione?
Solo la funzione!

Chi può usare i parametri formali di una funzione?
Solo la funzione

```
#include <stdio.h>
int main() {
    int primo, secondo;

    printf("...numeri... ");
    scanf("%d %d", &primo, &secondo);
```

`n1 = 2873;`

`result = 129;`

```
printf("mcd=%d\n", mcd(primo, secondo));
return 0;
}
```

non compila nemmeno ...

MEMORIA

primo 5

secondo 6

Record di Attivazione:
area di memoria riservata per l'esecuzione di una chiamata

mcd(primo, secondo)

AREA PARAMETRI

n1 5

n2 6

RDA

AREA VAR LOCALI

result ...

... ALTRE AREE (codice, punto di ritorno)

Ciclo di vita delle variabili e dei parametri

Esistono solo mentre la funzione è attivata (cioè mentre esiste il RDA che le contiene)

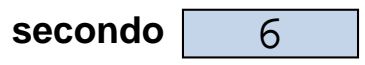
Quando la funzione termina, il RDA termina e le variabili locali e parametri non ci sono più (non sono più allocati in memoria)

```
#include <stdio.h>
int main() {
    int primo, secondo;

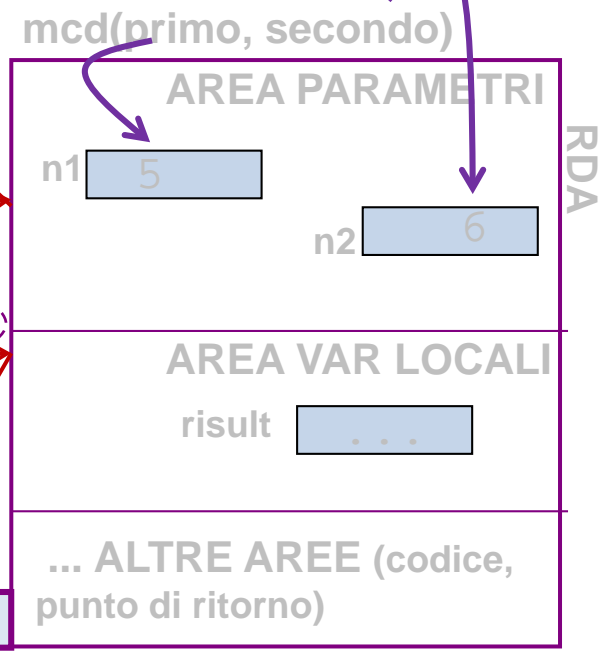
    printf("...numeri... ");
    scanf("%d %d", &n1, &n2);
    n1 = 2873;
    result = 129;
    printf("mcd=%d", mcd(primo, secondo));
    return 0;
}
```

non hanno senso ... n1 e result non esistono in questo punto del programma principale; e quando mcd viene chiamata esistono solo dentro al RDA ... protetti

MEMORIA



Record di Attivazione:
area di memoria riservata per l'esecuzione di una chiamata



Quanto detto vale per qualsiasi funzione

```
int mcd(int n1, int n2) {
    int result;

    secondo = 1000; /* tsk tsk ... */

    while (n1!=n2)
        if (n2>n1)n2=n2-n;
        else n1-=n2;
    result = n1;
    return result;
}

#include <stdio.h>
int main() {
    int primo, secondo;

    printf("...numeri... ");
    scanf("%d %d", &primo, &secondo);
    printf("mcd=%d\n", mcd(primo, secondo));
    return 0;
}
```

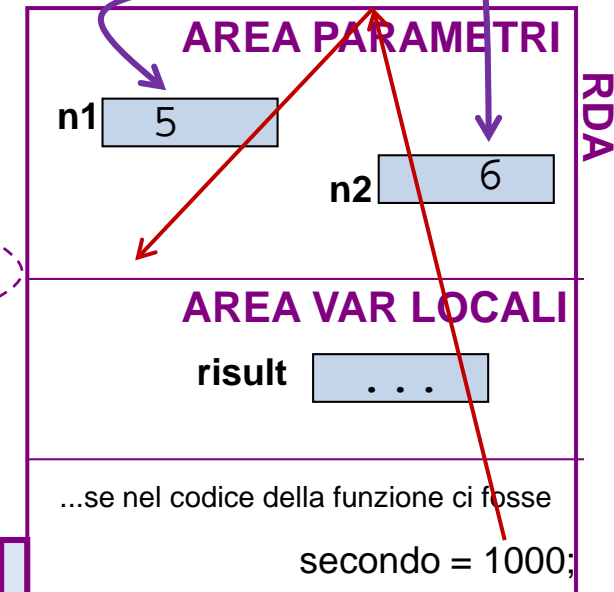
MEMORIA

primo 5

secondo 6

Record di Attivazione:
area di memoria riservata per
l'esecuzione di una chiamata

mcd(primo, secondo)



Stack delle chiamate di funzione

Quando una funzione ne chiama un'altra,

- la funzione chiamante rimane attivata, ma viene sospesa,
- la funzione chiamata viene attivata,
- e i RDA contengono lo stato di esecuzione delle funzioni attivate

ricordiamoci com'era il programma per stampare uno o molti 47 ...

- la funzione `main()` chiamava le altre funzioni, a seconda della scelta espressa dall'utente
- La funzione `stampaMolti47()` era chiamata da `main()`, e chiamava `fun47()`, quindi era contemporaneamente una *funzione chiamata* e una *funzione chiamante*;

```
void stampaMolti47 (int quanti47) {  
  
    for ( ; quanti47>0; quanti47--)  
        printf("sempre lui: %d\n", fun47());  
    }  
return;  
}
```

Stack delle chiamate: esempio

```
void menu();
int fun47 ();
int menoFun47 (void);
void stampaMolti47 (int );
```

```
int main () {
    int scelta, quanteVolte;
    printf(" Stimato/a utente, ...: \n");

    menu();
    scanf("%d", &scelta);
```

```
    if (scelta==1)
        printf("sempre lui: %d\n", fun47());
    else if (scelta==2)
        printf("ancora e sempre lui: %d\n", menoFun47());
    else if (scelta==3) {
        printf("ah! Dimmi quante volte ...: ");
        scanf("%d", &quanteVolte);
        stampaMolti47(quanteVolte);          /* questa chiamata stampa tante volte 47
                                             quanto vale la variabile quanteVolte */
    }
    else printf("come hai potuto? Scelta sbagliata. Addio.\n");
    printf("FINE\n");
    return 0;
}
```

```
Stimato/a utente, esegui una scelta scrivendo un numero intero
secondo quanto di seguito indicato:
--- Scrivi 1 per vedere il numero
--- Scrivi 2 per vedere il numero con una soluzione meno bella
--- Scrivi 3 per vedere il numero tante volte
    (Hai solo queste scelte ...)
3
ah! Dimmi quante volte vorresti veder stampato quel numero: 7
sempre lui: 47
sempre lui: 47
sempre lui: 47
sempre lui: 47
sempre lui: 47
sempre lui: 47
sempre lui: 47
FINE
```

Vediamo l'evoluzione dei RDA durante l'esecuzione del programma mostrata in figura

Stack delle chiamate di funzione 1/14

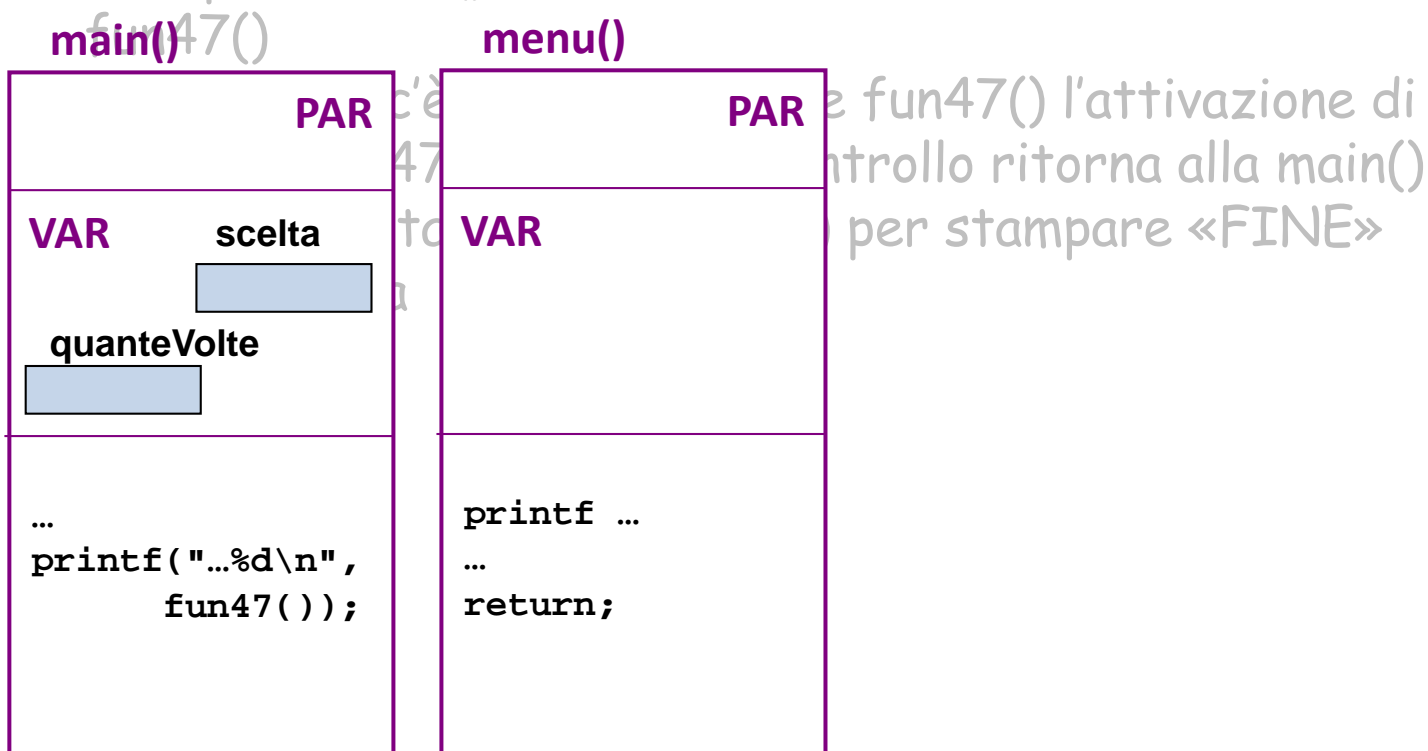
ricordiamoci com'era il programma per stampare uno o molti 47 ... fai avanti e indietro con la slide precedente ...

durante l'esecuzione mostrata nella figura precedente,

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione fun47()
- Quando non c'è più da chiamare fun47() l'attivazione di stampaMolti47 termina e il controllo ritorna alla main()
- che a sua volta chiama printf() per stampare «FINE»
- e poi termina

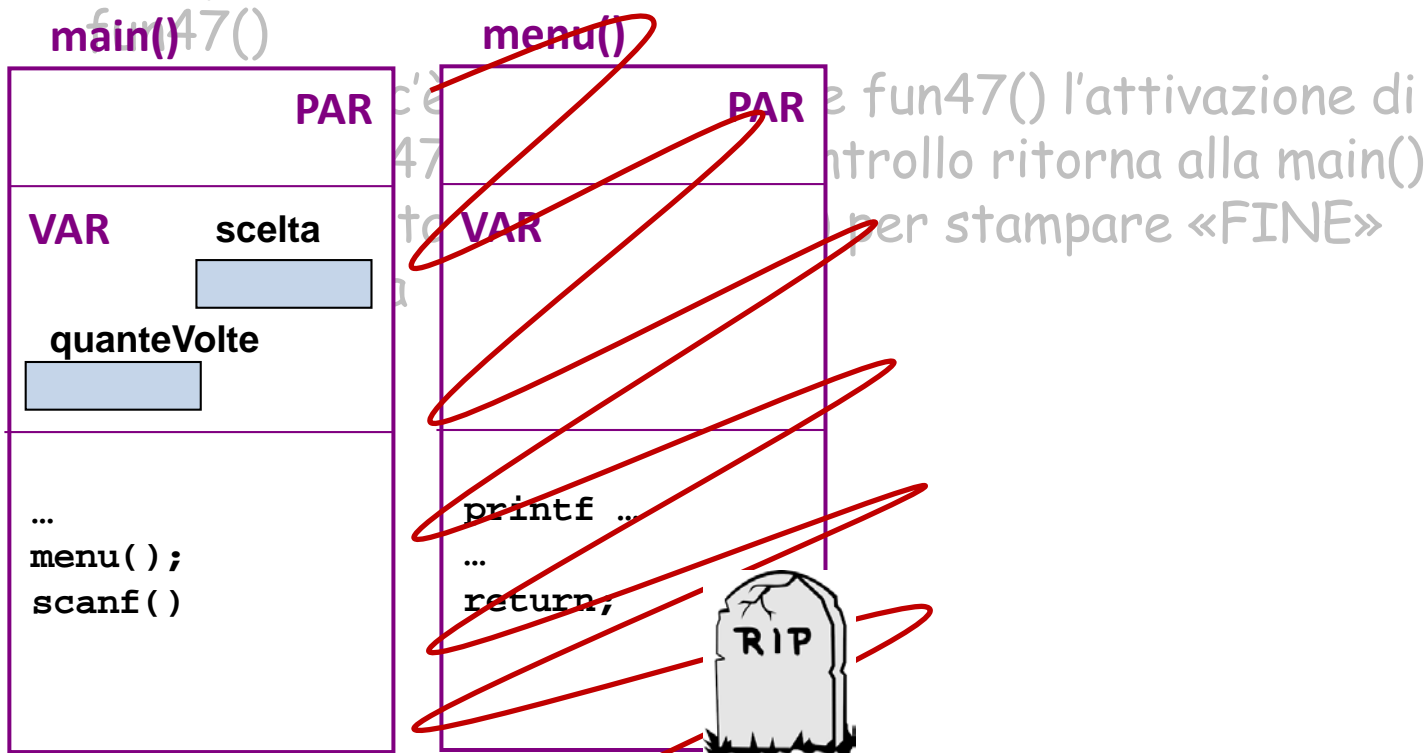
Stack delle chiamate di funzione 2/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione



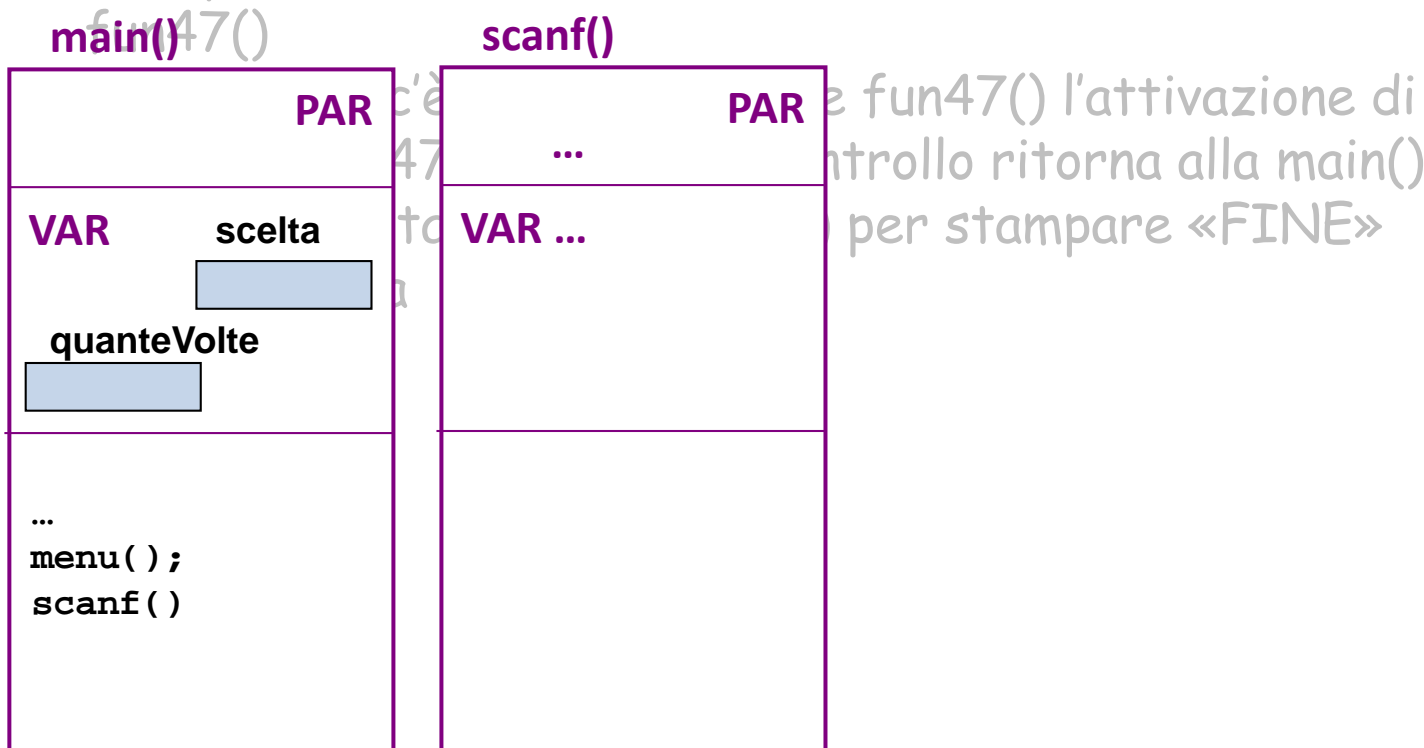
Stack delle chiamate di funzione 3.14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione



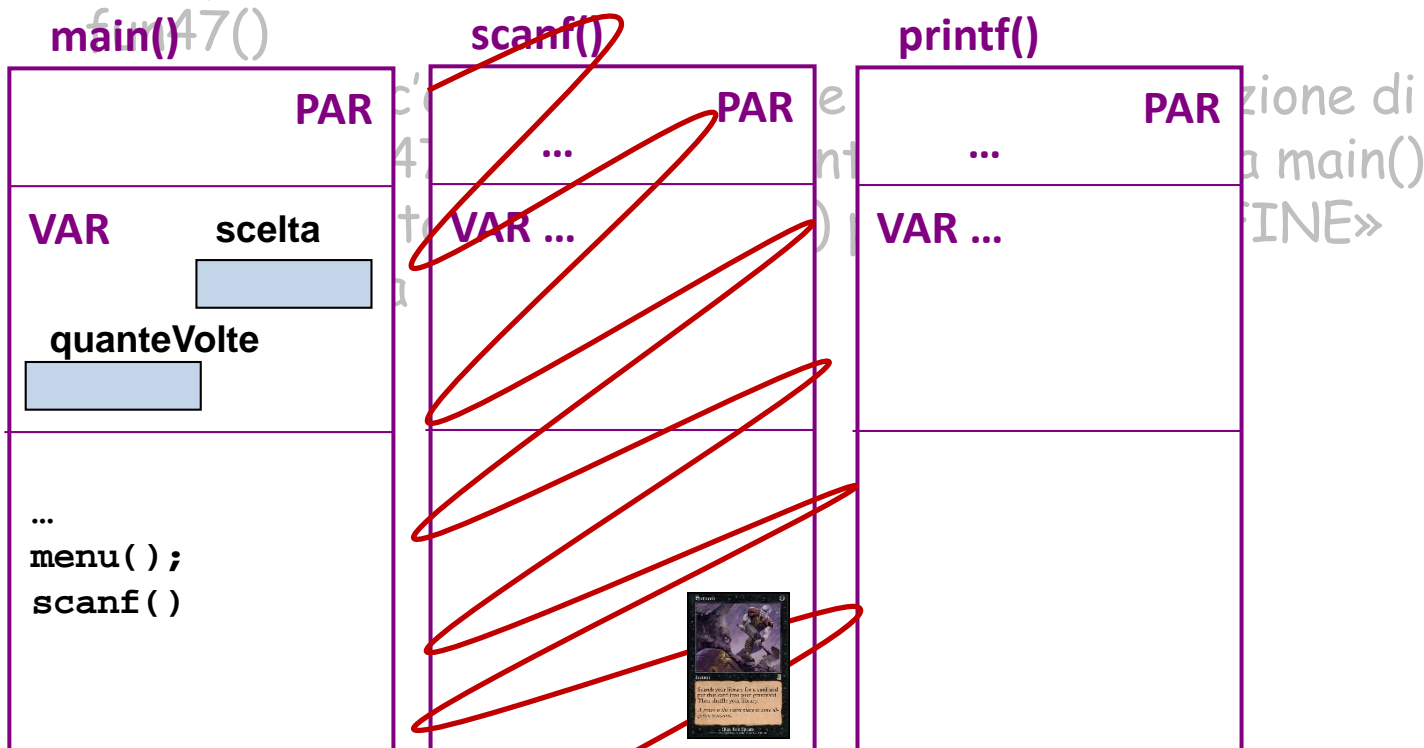
Stack delle chiamate di funzione 4/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione



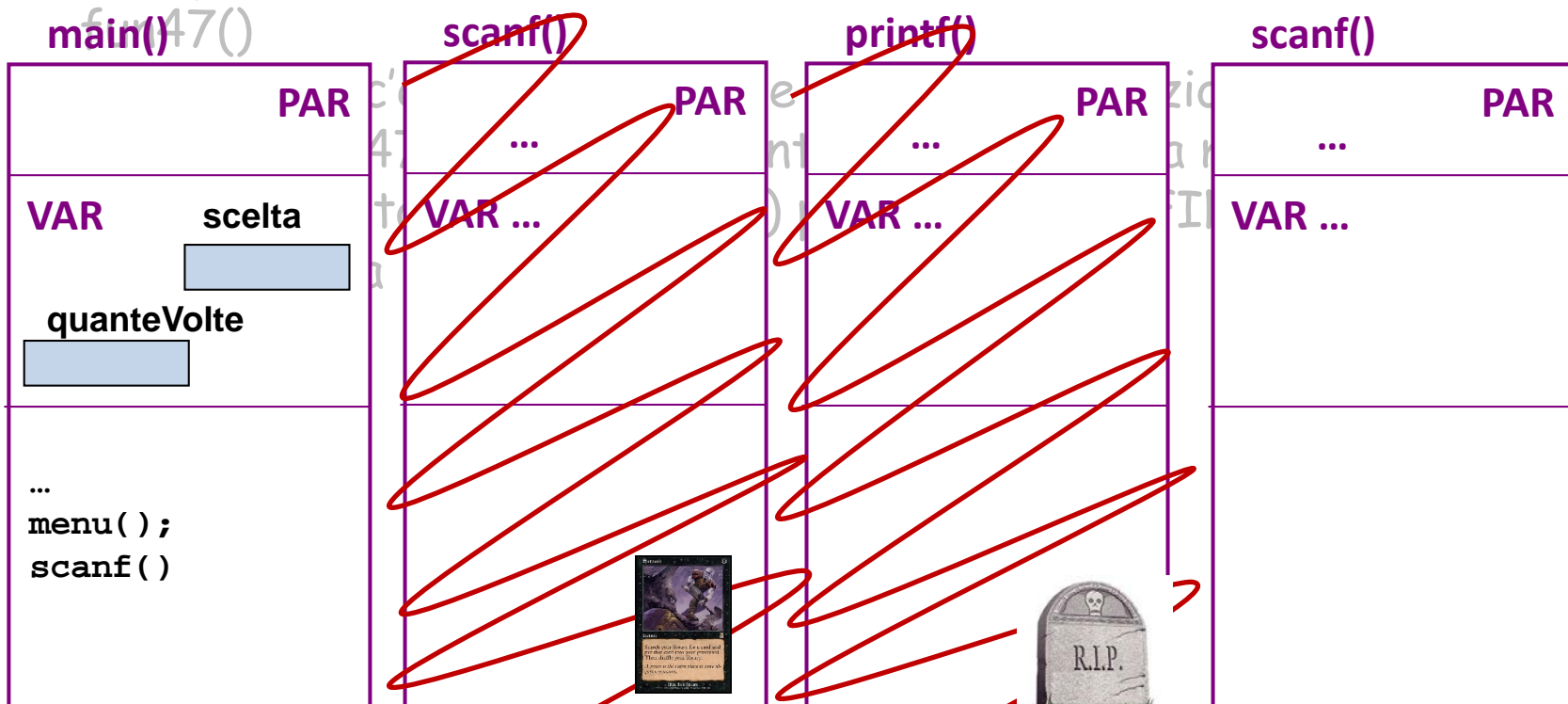
Stack delle chiamate di funzione 5/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione



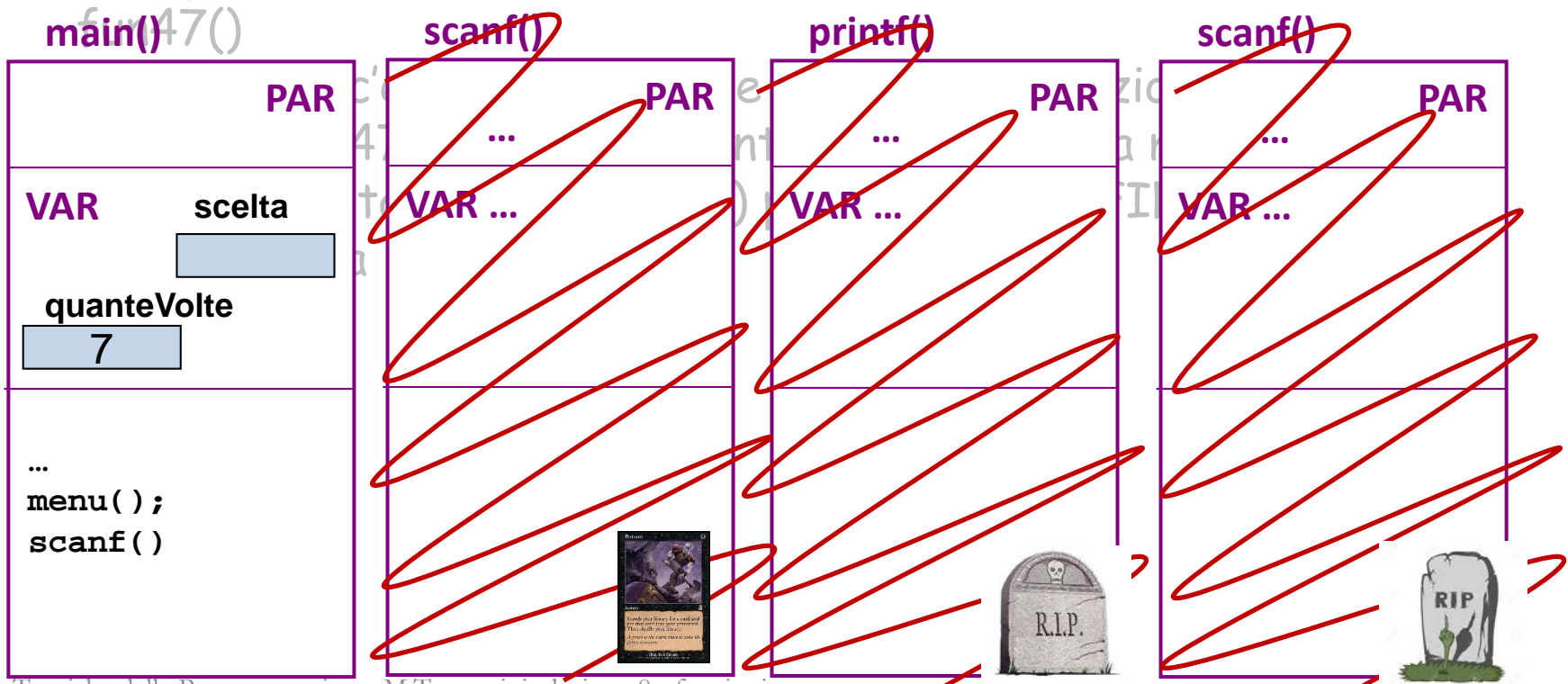
Stack delle chiamate di funzione 6/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione



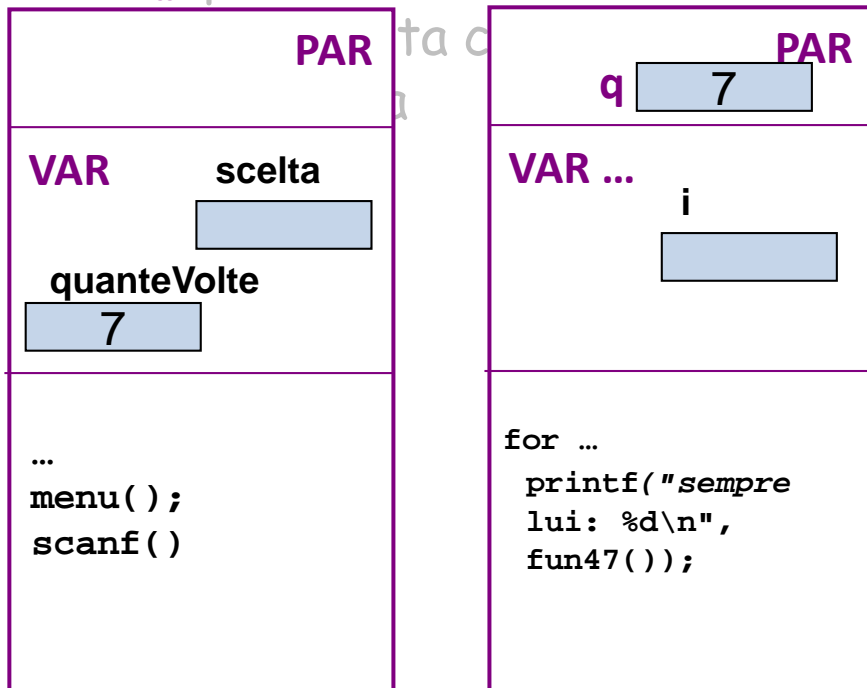
Stack delle chiamate di funzione 7/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione



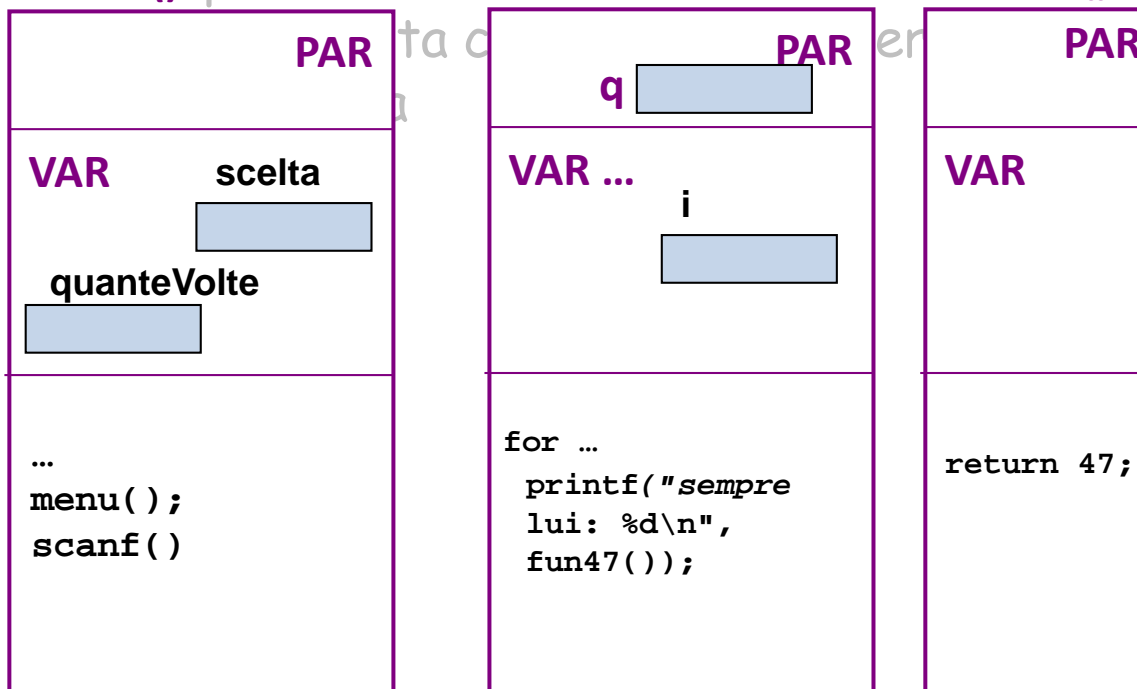
Stack delle chiamate di funzione 8/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama **stampaMolti47(quanteVolte)**
- stampaMolti47() a sua volta chiama - numerose volte: ne vediamo una sola - la funzione fun47()
- Quando non c'è più da chiamare fun47() l'attivazione di stampaMolti47 termina e il controllo ritorna alla main()



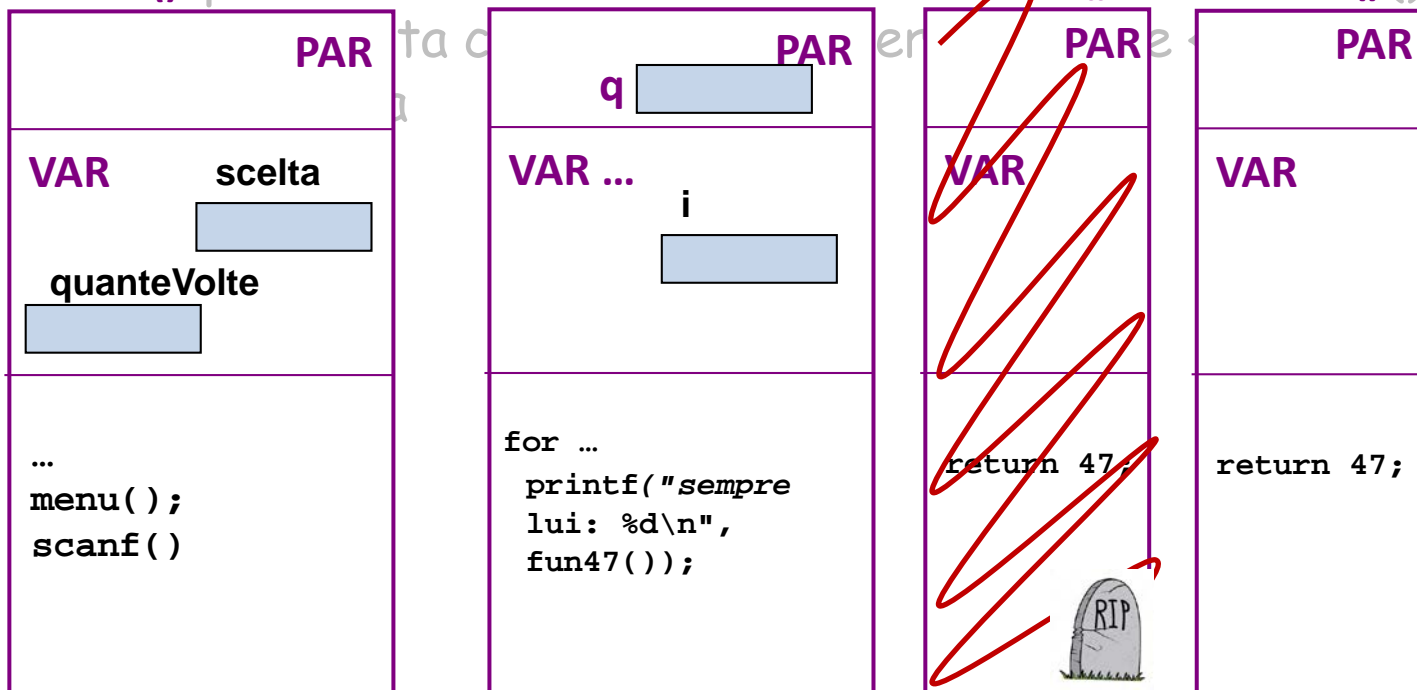
Stack delle chiamate di funzione 9/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama **stampaMolti47(quanteVolte)**
- stampaMolti47() a sua volta chiama - numerose volte - la funzione **fun47()**
- Quando non c'è più da chiamare fun47() l'attivazione di stampaMolti47 termina e il controllo ritorna alla main()



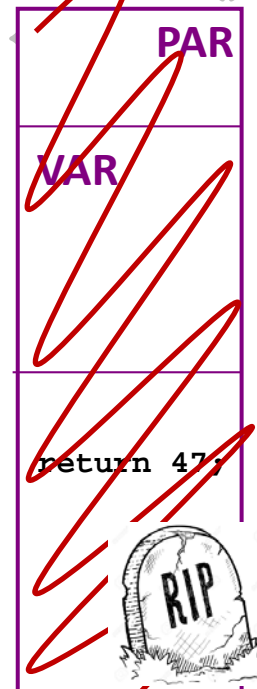
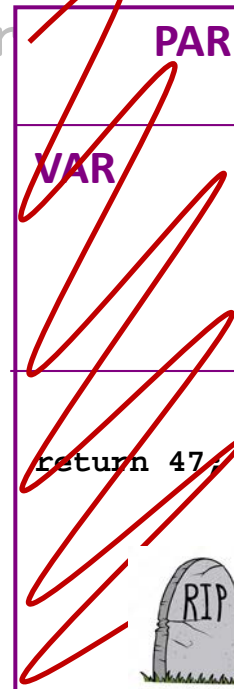
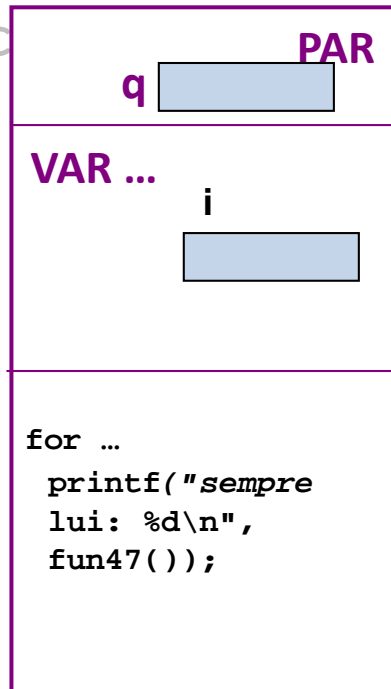
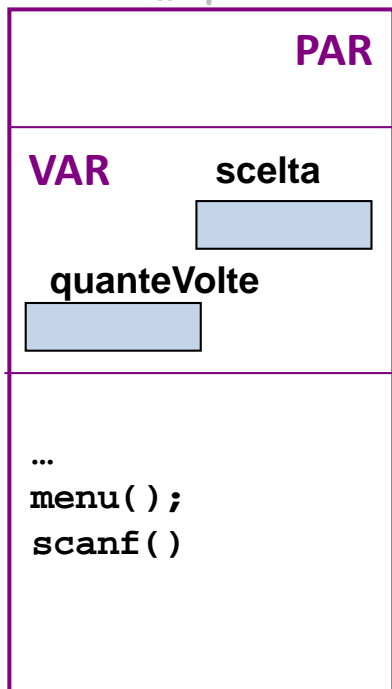
Stack delle chiamate di funzione 10/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama **stampaMolti47(quanteVolte)**
- stampaMolti47() a sua volta chiama - numerose volte - la funzione **fun47()**
- Quando non c'è più da chiamare fun47() l'attivazione di stampaMolti47() termina e il controllo ritorna alla main()



Stack delle chiamate di funzione 11/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama **stampaMolti47(quanteVolte)**
- stampaMolti47() a sua volta chiama - numerose volte - la funzione **fun47()**
- Quando non c'è più da chiamare fun47() l'attivazione di main() stampaMolti47() stampaMolti47(quanteVolte) fun47() fun47()

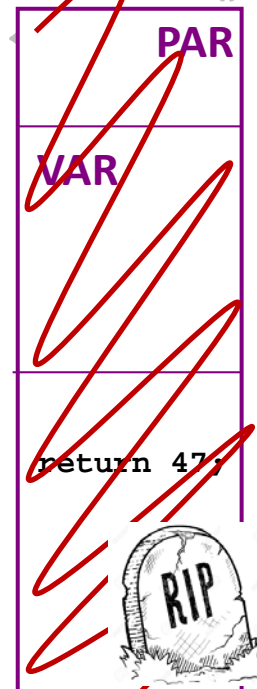
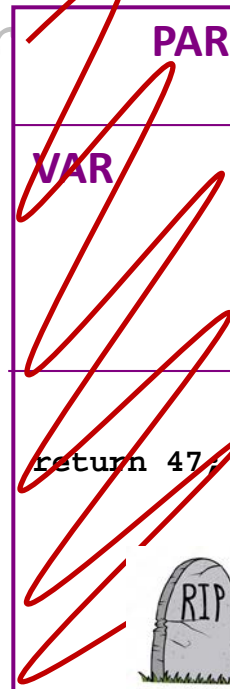
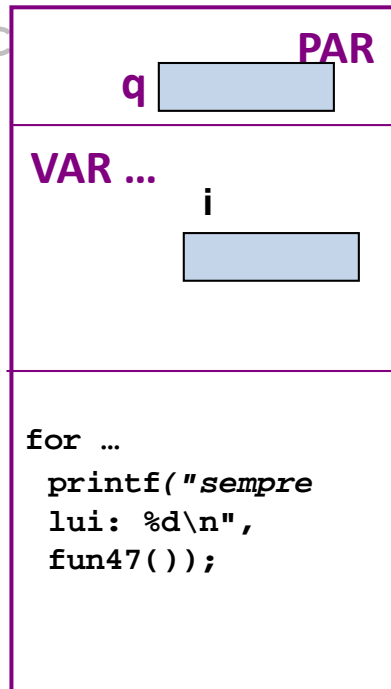
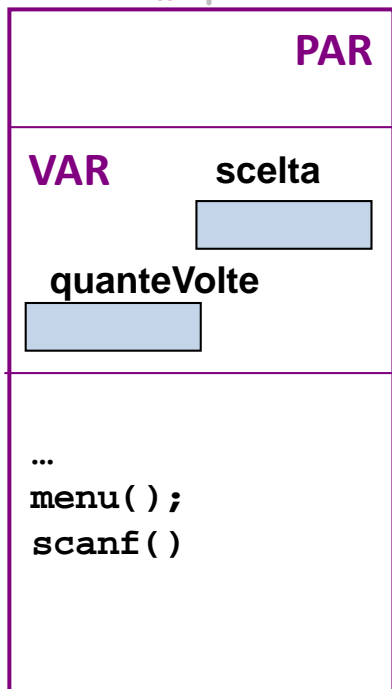


ok,
disegnare tutti
gli altri RDA
pertinenti
all'esecuzione
del programma

...

Stack delle chiamate di funzione 11/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama **stampaMolti47(quanteVolte)**
- stampaMolti47() a sua volta chiama - numerose volte - la funzione **fun47()**
- Quando non c'è più da chiamare fun47() l'attivazione di main() stampaMolti47() stampaMolti47(quanteVolte) fun47() fun47()

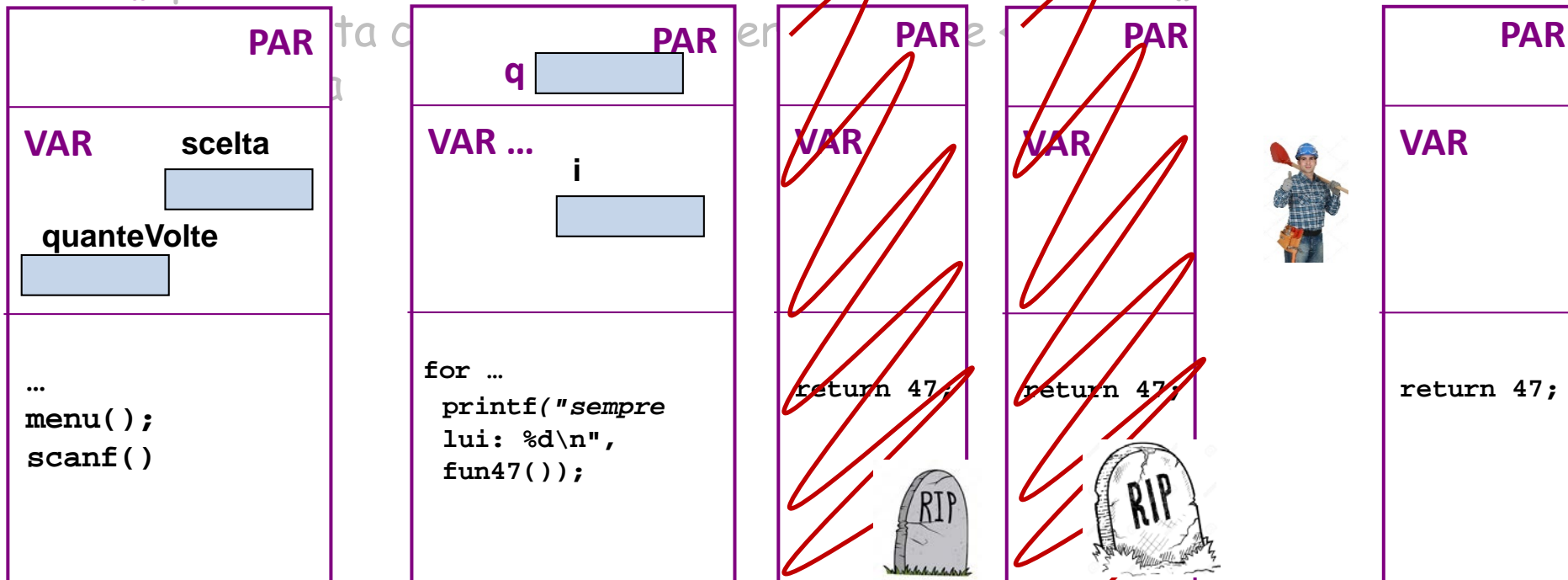


altri 4
RDA per
fun47()
... uno per uno vengono
allocati, eseguiti e poi
deallocati.
E poi un altro, il settimo
... e poi il resto ...



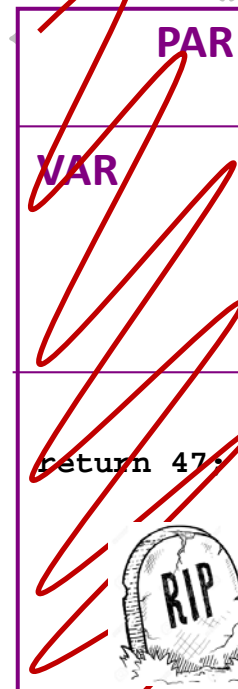
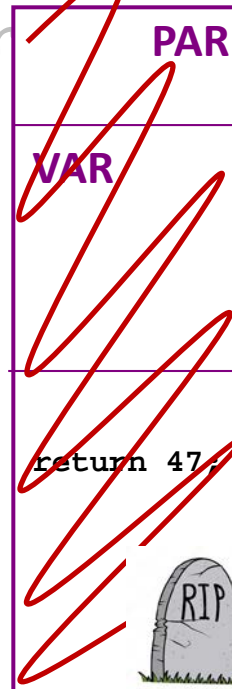
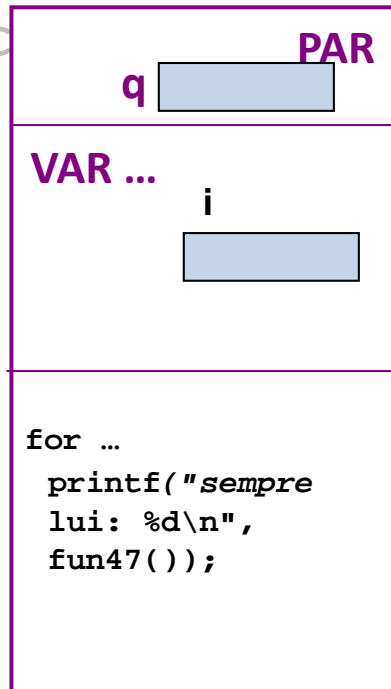
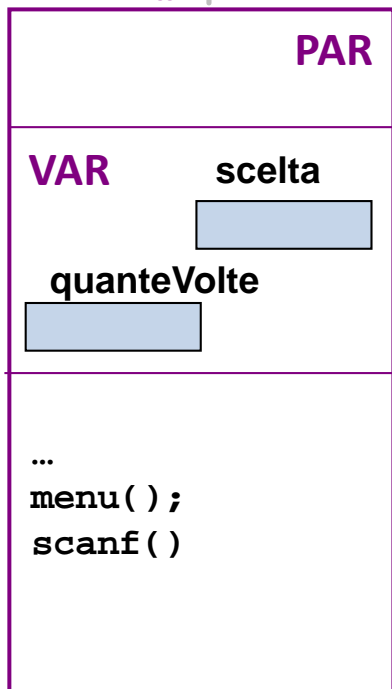
Stack delle chiamate di funzione 11/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama **stampaMolti47(quanteVolte)**
- stampaMolti47() a sua volta chiama - numerose volte - la funzione **fun47()**
- Quando non c'è più da chiamare fun47() l'attivazione di



Stack delle chiamate di funzione 11/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama **stampaMolti47(quanteVolte)**
- stampaMolti47() a sua volta chiama - numerose volte - la funzione **fun47()**
- Quando non c'è più da chiamare fun47() l'attivazione di

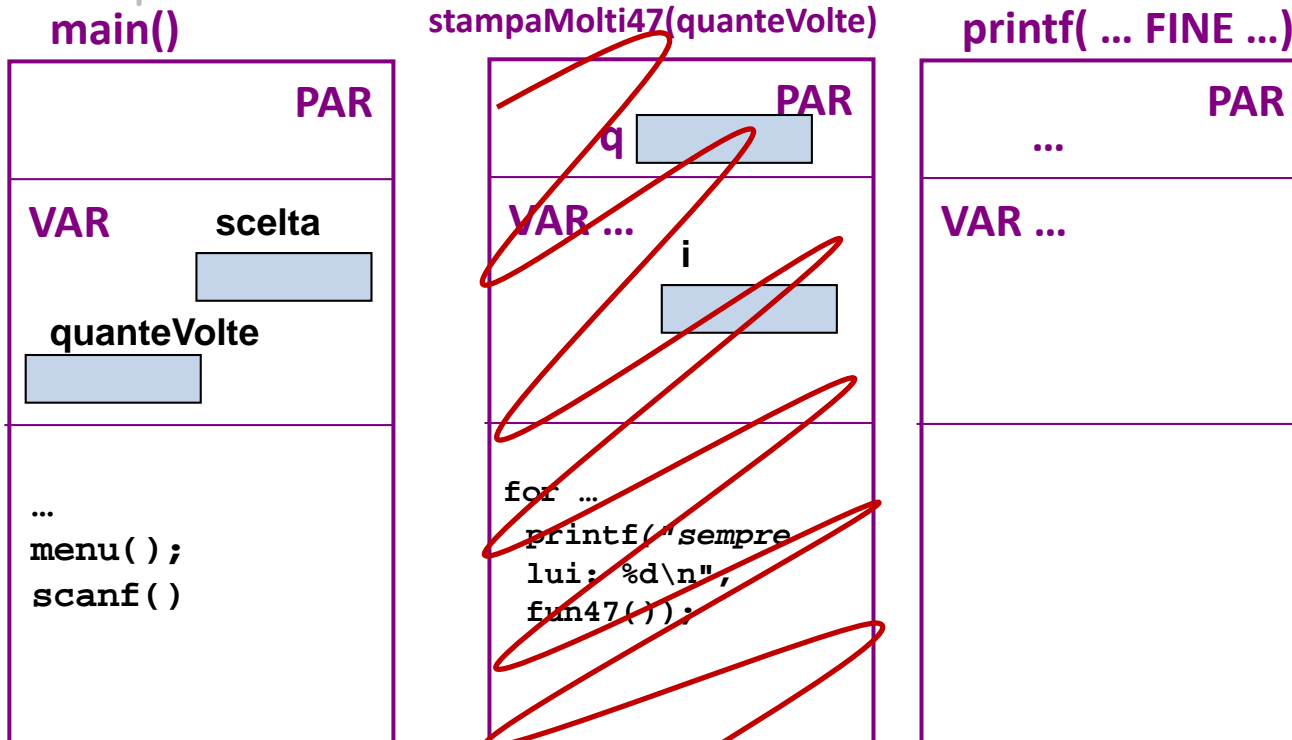


...e poi il resto ...



Stack delle chiamate di funzione 12/14

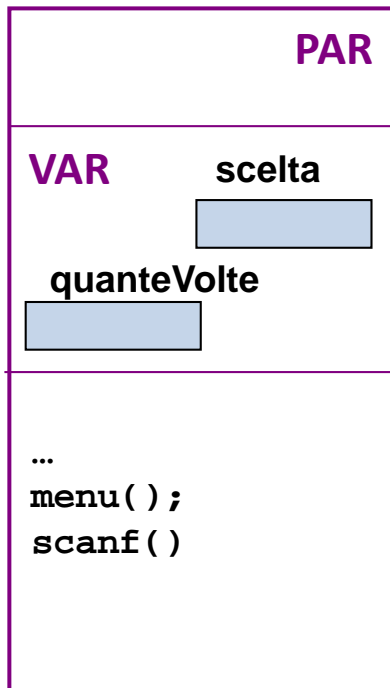
- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione fun47()
- Quando non c'è più da chiamare fun47() l'attivazione di stampaMolti47 termina e il controllo ritorna alla main()
- Che a sua volta chiama printf() per stampare «FINE»
- E poi termina



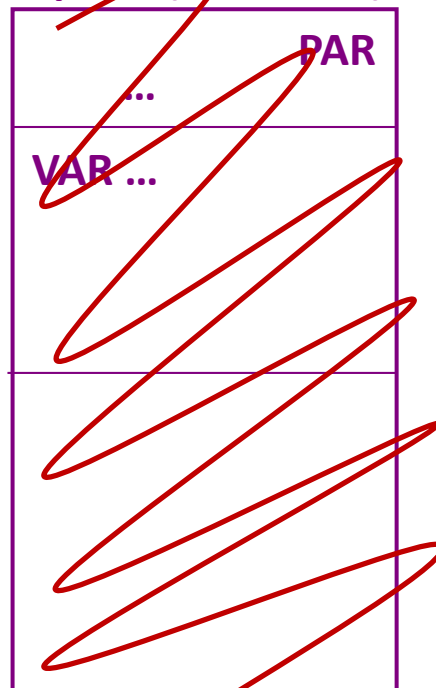
Stack delle chiamate di funzione 13/14

- è attiva main() ovviamente, che chiama menu()
- poi menu() termina, avendo stampato il menù
- poi viene chiamata scanf() per leggere la scelta
- al termine di scanf() la main riprende, e chiama printf()
- e poi, al termine di printf(), di nuovo scanf()
- Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
- stampaMolti47() a sua volta chiama - numerose volte - la funzione fun47()
- Quando non c'è più da chiamare fun47() l'attivazione di stampaMolti47 termina e il controllo ritorna alla main()
- Che a sua volta chiama printf() per stampare «FINE»
- E poi termina

main()



printf(... FINE ...)



Stack delle chiamate di funzione 14/14

- è attiva main() ovviamente, che chiama menu()
 - poi menu() termina, avendo stampato il menù
 - poi viene chiamata scanf() per leggere la scelta
 - al termine di scanf() la main riprende, e chiama printf()
 - e poi, al termine di printf(), di nuovo scanf()
 - Al termine di scanf() abbiamo anche il dato in quanteVolte e la main() chiama stampaMolti47(quanteVolte)
 - stampaMolti47() a sua volta chiama - numerose volte - la funzione fun47()
 - Quando non c'è più da chiamare fun47() l'attivazione di stampaMolti47 termina e il controllo ritorna alla main()
 - Che a sua volta chiama printf() per stampare «FINE»
- E poi anche main(), cioè il programma, termina**

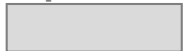
main()

PAR

VAR scelta



quanteVolte



...
menu () ;
scanf ()



Osservazioni

Come progettare una funzione

- algoritmo ...
- quali parametri? Solo quelli indispensabili: i dati che occorrono alla funzione per fare il proprio lavoro

double potenza (double numero, int esponente)



int mcd (int n1, int n2)



double areaCerchio (double raggio, double pi)



Osservazioni

Come progettare una funzione

- algoritmo ...
- quali parametri? Solo quelli indispensabili: i dati che occorrono alla funzione per fare il proprio lavoro

double potenza (double numero, int esponente) OK

int mcd (int n1, int n2) OK

double areaCerchio (double raggio, double pi) KO

RESTITUIRE?

Osservazioni

RESTITUIRE

- è quel che fa la funzione al termine (a meno che non sia void)
- fornisce (*restituisce*, *ritorna*), alla funzione chiamante, il valore che ha calcolato durante l'esecuzione. Ora la funzione chiamante può usare quel valore per i suoi successivi calcoli

ad esempio, la main() chiama potenza()

```
potenzaVal = potenza (n, e)
```

e ora ha il valore risultante dal calcolo e può usarlo (ad esempio stamparlo)

```
printf("...%g...", potenzaVal )
```

OK

```
int mcd(int n1, int n2) {  
    int result;  
    while (n1!=n2)  
        if (n2>n1)  
            n2=n2-n;  
        else n1-=n2;  
    result = n1;  
    printf("il risultato è %d", result);  
    return 0;  
}
```

KO

perche' ?