



Funzioni

Fondamenti di Informatica, AA 2022/23
Luca Cassano

luca.cassano@polimi.it



A cosa servono funzioni?

Calcolo del **fattoriale**

Scrivere un programma che legge da tastiera un intero x e calcola $fattX = \prod_{i=1}^x i$

Se $fattX$ è maggiore di 220, il programma legge da tastiera un intero y e calcola $fattY = \prod_{i=1}^y i$



A cosa servono funzioni?

```
int i, fattX, fattY, x, y;  
scanf("%d", &x);  
fattX = 1;  
for(i = 1; i <= x; i++)  
    fattX = fattX * i;
```



A cosa servono funzioni?

```
int i, fattX, fattY, x, y;
scanf("%d", &x);
fattX = 1;
for(i = 1; i <= x; i++)
    fattX = fattX * i;

if(fattX > 220) {

}
```



A cosa servono funzioni?

```
int i, fattX, fattY, x, y;
scanf("%d",&x);
fattX = 1;
for(i = 1; i <= x; i++)
    fattX = fattX * i;

if(fattX > 220) {
    scanf("%d",&y);
    fattY = 1;
    for(i = 1; i <= y; i++)
        fattY = fattY * i;
}
```



A cosa servono funzioni?

```
int i, fattX, fattY, x, y;
```

```
scanf("%d", &x);  
fattX = 1;  
for(i = 1; i <= x; i++)  
    fattX = fattX * i;
```

```
if(fattX > 220) {
```

```
    scanf("%d", &y);  
    fattY = 1;  
    for(i = 1; i <= y; i++)  
        fattY = fattY * i;
```

```
}
```

Entrambi i
frammenti di
codice
eseguono il
calcolo del
fattoriale



A cosa servono funzioni?

Riusabilità

- Scrivo una sola volta codice utilizzato spesso
- Lo stesso codice viene richiamato in diversi programmi



A cosa servono funzioni?

Riusabilità

- Scrivo una sola volta codice utilizzato spesso
- Lo stesso codice viene richiamato in diversi programmi

Leggibilità

- Incapsulo porzioni di codice complesso, il programmatore non deve entrare nei dettagli



A cosa servono funzioni?

Flessibilità

- Posso aggiungere funzionalità non presenti nelle funzioni di libreria



A cosa servono funzioni?

Flessibilità

- Posso aggiungere funzionalità non presenti nelle funzioni di libreria

Manutenibilità

- Modifiche e correzioni sono gestibili facilmente
- E' più difficili commettere errori sistematici



Le funzioni

```
int fatt(int x) {  
    int fattX = 1;  
    for(i = 1; i <= x; i++)  
        fattX = fattX * i;  
    return fattX;  
}
```



Le funzioni

```
int fatt(int x) {  
    int fattX = 1;  
    for (i = 1; i <= x; i++)  
        fattX = fattX * i;  
    return fattX;  
}
```

x è l'argomento
della funzione (serve
a fornire l'input)



Le funzioni

```
int fatt(int x) {  
    int fattX = 1;  
    for(i = 1; i <= x; i++)  
        fattX = fattX * i;  
    return fattX;  
}
```

x è l'**argomento** della funzione (serve a fornire l'input)

fattX è il **valore di ritorno** della funzione (serve a fornire l'output)



Le funzioni

```
int fatt(int x) {  
    int fattX = 1;  
    for (i = 1; i <= x; i++)  
        fattX = fattX * i;  
    return fattX;  
}
```

The first line of the code, `int fatt(int x)`, is bracketed in red and labeled **header** in red text.

x è l'**argomento** della funzione (serve a fornire l'input)

fattX è il **valore di ritorno** della funzione (serve a fornire l'output)

- La testata (o intestazione o header o prototipo) dichiara:
 - nome della funzione
 - argomenti e tipi degli argomenti(input)
 - Tipo del valore di ritorno (output)



Le funzioni

```
int fatt(int x) {  
    int fattX = 1;  
    for (i = 1; i <= x; i++)  
        fattX = fattX * i;  
    return fattX;  
}
```

header

body

x è l'**argomento** della funzione (serve a fornire l'input)

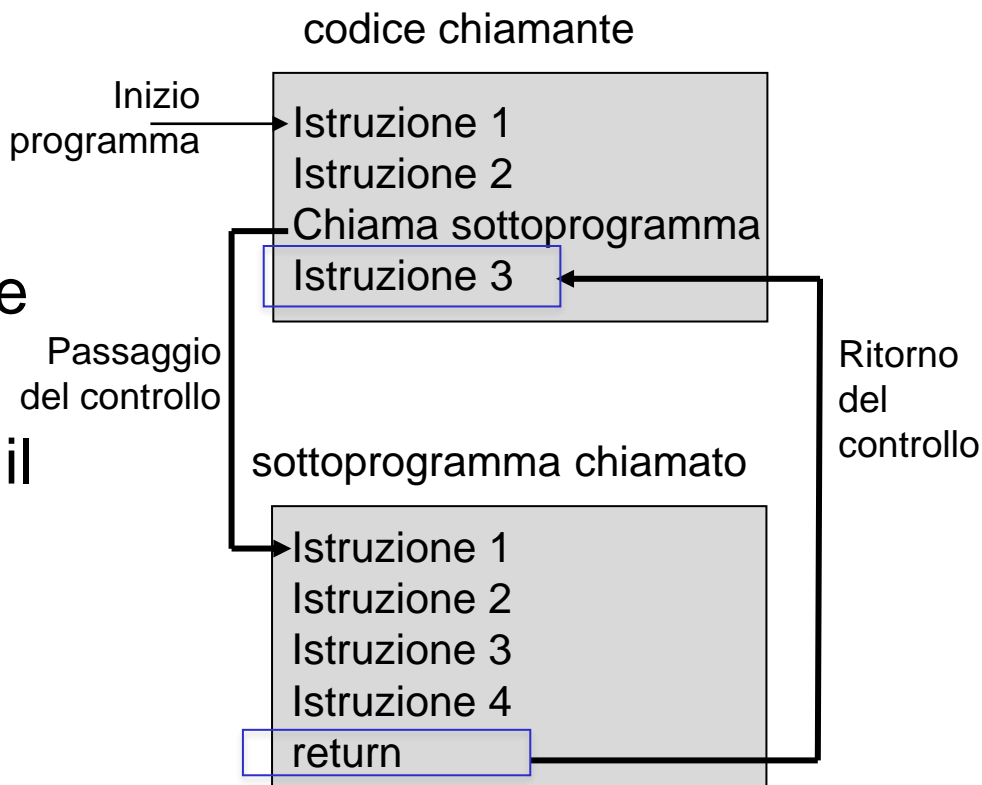
fattX è il **valore di ritorno** della funzione (serve a fornire l'output)

- La testata (o intestazione o header o prototipo) dichiara:
 - nome della funzione
 - argomenti e tipi degli argomenti(input)
 - Tipo del valore di ritorno (output)
- Il corpo definisce le istruzioni da eseguire quando la funzione viene invocata
 - Utilizza gli argomenti e definisce il valore di ritorno



Invocazione di una funzione

- All'atto della chiamata, il controllo dell'esecuzione passa dal chiamante al chiamato
- Il codice del chiamato viene eseguito
- Al termine dell'esecuzione il controllo ritorna al chiamante, all'istruzione successiva a quella della chiamata





Sintassi per la Definizione di una Funzione

La sintassi per la definizione di una funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```



Sintassi per la Definizione di una Funzione

La sintassi per la definizione di una funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```

NB: posso dichiarare il prototipo della funzione in un punto del programma e definirne il corpo in un altro punto

NB: è obbligatorio che il prototipo di una funzione sia dichiarato prima del suo utilizzo (il corpo può essere definito anche dopo)



Sintassi per la Definizione di una Funzione

La sintassi per la definizione di una funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```

return è l'ultima istruzione di un sottoprogramma indica

- Il valore restituito
- Il punto in cui il controllo torna al chiamante (si esce dal sottoprogramma)
- Dopo l'esecuzione di **return** verrà ripristinata l'esecuzione del chiamante dal punto successivo alla chiamata



Sintassi per la Definizione di una Funzione

La sintassi per la definizione di una funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```

In un sottoprogramma

- Deve esserci almeno un'istruzione di **return**
- Possono esserci più istruzioni di **return** ma in alternativa
- Il sottoprogramma può restituire un solo valore (NON un array!)



Sintassi per la Definizione di una Funzione

La sintassi per la definizione di una funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```

Il valore restituito da una funzione chiamata verrà (generalmente) assegnato (e sarà quindi un right value) a una variabile dichiarata dal chiamante



Sintassi per la Definizione di una Funzione

La sintassi per la definizione di una funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```

NB: una funzione senza parametri di ingresso è definita come:

```
tipoOut nomeFunz() {  
    <corpo della funzione>  
    return outParam;  
}
```



Sintassi per la Definizione di una Funzione

La sintassi per la definizione di una funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```

NB: una funzione senza parametri di ingresso è definita come:

```
tipoOut nomeFunz()  
{  
    <corpo della funzione>  
    return outParam;  
}
```

Nessun
parametro
specificato



Sintassi per la Definizione di una Funzione

La sintassi per una funzione funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, .., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```

NB: una funzione senza parametri di uscita è definita come:

```
void nomeFunz(tipo1 inParam1, .., tipoN inParamN) {  
    <corpo della funzione>  
}
```




Sintassi per la Definizione di una Funzione

La sintassi per una funzione funzione è:

```
tipoOut nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
    return outParam;  
}
```

NB: una funzione senza parametri di uscita è definita come:

```
void nomeFunz(tipo1 inParam1, ..., tipoN inParamN) {  
    <corpo della funzione>  
}
```

«tipo del risultato» **void**

Nessun **return**



Invocazione di una funzione

Una funzione può essere invocata in un programma attraverso il suo nome, seguito dagli argomenti fra parentesi rotonde

La funzione viene quindi eseguita e il suo valore di ritorno viene calcolato



```
int fatt(int x) {  
    fattX = 1;  
    for(i = 1; i <= x; i++)  
        fattX = fattX * i;  
    return fattX;  
}
```



Invocazione di una funzione

Una funzione può essere invocata in un programma attraverso il suo nome, seguito dagli argomenti fra parentesi rotonde

La funzione viene quindi eseguita e il suo valore di ritorno viene calcolato

```
scanf("%d", &x);  
fattX = fatt(x);   
if (fattX > 220) {  
    scanf("%d", &y);  
    fattY = fatt(y);   
}
```

```
int fatt(int x) {  
    fattX = 1;  
    for(i = 1; i <= x; i++)  
        fattX = fattX * i;  
    return fattX;  
}
```



Invocazione di una funzione

Una funzione può essere invocata in un programma attraverso il suo nome, seguito dagli argomenti fra parentesi rotonde.

La funzione
calcola

viene

Se la funzione è di tipo void non si sarà nessun valore restituito e quindi nessun assegnamento all'atto della sua invocazione

```
void stampaSomma(int a, int b) {  
    printf("%d", a+b);  
}
```

sc

fa

if

```
/* una possibile invocazione */  
stampaSomma(5, 22);
```

l++)

i;

```
fattY = fatt(y);
```



```
}
```



Definizioni:

- I **parametri formali** sono le variabili usate come argomenti **nella definizione** della funzione



Definizioni:

- I **parametri formali** sono le variabili usate come argomenti **nella definizione** della funzione
- I **parametri attuali** sono i valori (o le variabili) usati come argomenti **nell'invocazione** della funzione



Definizioni:

```
/* definizione della funzione */
```

```
int fatt(int x) {
```

```
    fattX = 1;
```

x è parametro formale

```
    for (i = 1; i <= x; i++)
```

```
        fattX = fattX * i;
```

```
    return fattX;
```

```
}
```



Definizioni:

/ definizione della funzione */*

```
int fatt(int x) {  
    fattX = 1;  
    for (i = 1; i <= x; i++)  
        fattX = fattX * i;  
    return fattX;  
}
```

x è parametro formale

/ invocazione della funzione */*

```
scanf("%d", &x);  
fattX = fatt(x);
```

x è parametro attuale



I Parametri (2)

Sono ammessi parametri scalari di qualsiasi tipo e strutture; no array!!!



I Parametri (2)

Sono ammessi parametri scalari di qualsiasi tipo e strutture; **no array!!!**

Approfondiremo in seguito il passaggio dei parametri



I Parametri (2)

Sono ammessi parametri scalari di qualsiasi tipo e strutture; **no array!!!**

I parametri attuali vengono **associati a quelli formali** in base alla **posizione**:

Esempio
`s = somma(4,5)`

```
int somma(a,b) {  
    return a+b;  
}
```



I Parametri (2)

Sono ammessi parametri scalari di qualsiasi tipo e strutture; **no array!!!**

I parametri attuali vengono **associati a quelli formali** in base alla **posizione**:

- il primo parametro attuale viene associato al primo formale

Esempio

s = somma(4,5)

```
int somma(a,b) {  
    return a+b;  
}
```



I Parametri (2)

Sono ammessi parametri scalari di qualsiasi tipo e strutture; **no array!!!**

I parametri attuali vengono **associati a quelli formali** in base alla **posizione**:

- il primo parametro attuale viene associato al primo formale
- il secondo parametro attuale al secondo parametro formale

Esempio

s = somma(4,5)

```
int somma(a,b) {  
    return a+b;  
}
```



I Parametri (2)

Sono ammessi parametri scalari di qualsiasi tipo e strutture; **no array!!!**

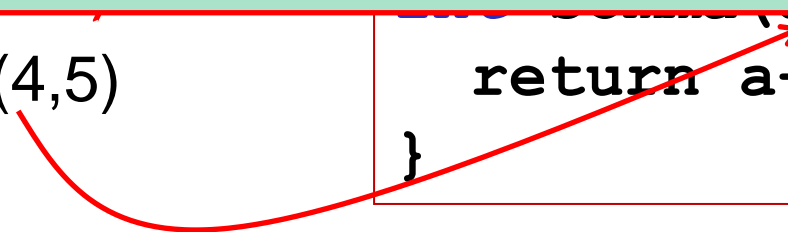
I parametri attuali vengono **associati a quelli formali** in base a

- associato al primo
- Il numero e il tipo di parametri di ingresso **attuali** all'invocazione della funzione deve essere identico al numero e al tipo di **parametri di ingresso formali della funzione** lo parametro

Esempio

`s = somma(4,5)`

```
int somma(int a, int b) {  
    return a+b;  
}
```





I Parametri (2)

Sono ammessi parametri scalari di qualsiasi tipo e strutture; **no array!!!**

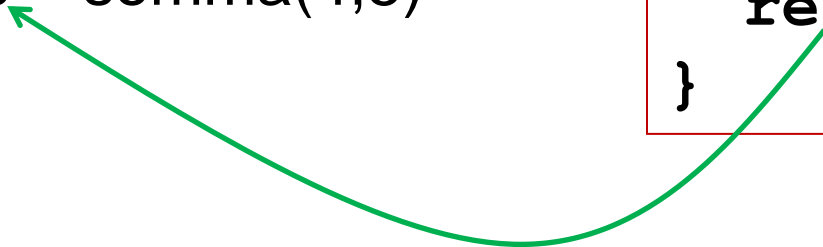
I parametri attuali vengono **associati a quelli formali** in base alla **posizione**:

- il primo parametro attuale viene associato al primo formale
- il secondo parametro attuale al secondo parametro formale

Esempio

s = somma(4,5)

```
int somma(a,b) {  
    return a+b;  
}
```





Invocazione di una funzione

Cosa fa questa funzione?

```
int seq_max(int N)
{
    int i,x,max;
    scanf("%d",&max);
    for(i=1;i<N;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
    }
    return max;
}
```




Invocazione di una funzione

Ogni invocazione di una funzione ha una vita indipendente

```
int main()
{
    int sum_max;
    sum_max = seq_max(10);
    sum_max = sum_max + seq_max(5);
    return 0;
}
```

```
int seq_max(int N)
{
    int i,x,max;
    scanf("%d",&max);
    for(i=1;i<N;i++) {
        scanf("%d",&x);
        if (x>max) max = x;
    }
    return max;
}
```



Visibilità delle variabili

```
void f(int p);  
  
int main()  
{  
    int a, b;  
    for (a=0; a<5; a++)  
    {  
    }  
}  
  
void f(int p)  
{  
    int c;  
}
```

In quali parti del programma sono **visibili** le variabili a, b, c e p?



Visibilità delle variabili

```
void f(int p);  
  
int main()  
{  
    int a, b;  
    for (a=0; a<5; a++)  
    {  
    }  
}  
  
void f(int p)  
{  
    int c;  
}
```

In quali parti del programma sono **visibili** le variabili a, b, c e p?

Regole di visibilità (regole di scope) :

- le variabili sono visibili (ed accessibili) solo nella funzione che le ha dichiarate



Visibilità delle variabili

```
void f(int p);  
  
int main()  
{  
    int a, b;  
    for (a=0; a<5; a++)  
    {  
    }  
}  
  
void f(int p)  
{  
    int c;  
}
```

In quali parti del programma sono **visibili** le variabili a, b, c e p?

Regole di visibilità (regole di scope) :

- le variabili sono visibili (ed accessibili) solo nella funzione che le ha dichiarate
- Variabili dichiarate in funzioni diverse possono avere lo stesso nome

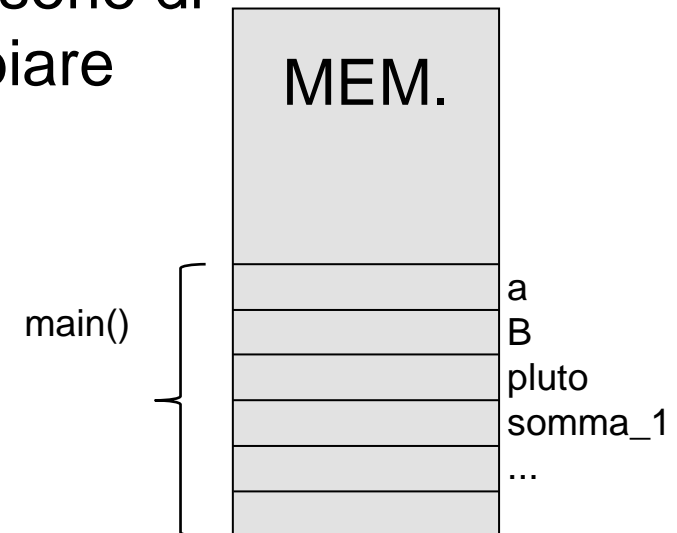


Il record di attivazione

Per ora abbiamo visto che tutte le variabili definite nel `main` sono collocate in celle adiacenti della memoria

In generale la gestione della memoria è molto più complessa e richiede ulteriori meccanismi per gestire l'esecuzione dei sottoprogrammi

- In base al flusso di esecuzione ciascun sottoprogramma può essere chiamato più volte e la serie di sottoprogrammi chiamati può cambiare





Il record di attivazione

Ad ogni invocazione di sottoprogramma viene riservata un'area di memoria chiamata “record di attivazione”

- Il record di attivazione contiene le variabili e i parametri formali a cui verrà assegnato il valore dei parametri attuali

Il record di attivazione di un sottoprogramma viene creato al momento della chiamata e rilasciato quando il sottoprogramma termina

In una sequenza di chiamate, l'ultima chiamata è la prima a terminare



Il record di attivazione

La zona di memoria che contiene i record di attivazione di un sottoprogramma è gestito con la logica di una pila (stack in inglese)

- L'ultimo elemento inserito nella stack è il primo ad essere estratto
- Logica LIFO (Last In First Out)

La dimensione del record di attivazione è già nota in fase di compilazione

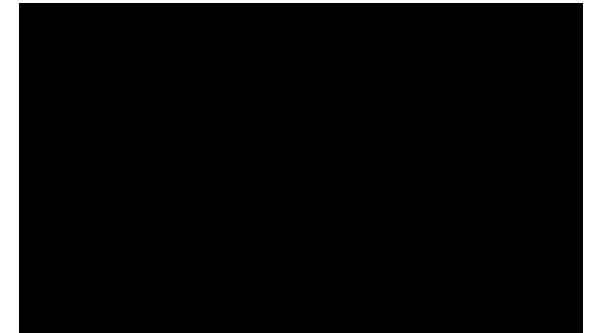
Il numero di attivazioni del sottoprogramma non è noto

Il primo record di attivazione è destinato al `main()`



Il record di attivazione

```
void a();
void b();
void c();
int main(){
    ...
    a();
    ...
}
void a(){
    printf("Esecuzione di a\n");
    b();
    printf("Termine di a\n");
}
void b(){
    printf("Esecuzione di b\n");
    c();
    printf("Termine di b\n");
}
void c(){
    printf("Esecuzione di c\n");
    ...
    printf("Termine di c\n");
}
```



TERMINALE



MEMORIA



Il record di attivazione

```
void a();
void b();
void c();
int main(){
    ...
    a();
    ...
}
void a(){
    printf("Esecuzione di a\n");
    b();
    printf("Termine di a\n");
}
void b(){
    printf("Esecuzione di b\n");
    c();
    printf("Termine di b\n");
}
void c(){
    printf("Esecuzione di c\n");
    ...
    printf("Termine di c\n");
}
```

Esecuzione di a

TERMINALE

Record di a

Record di main

MEMORIA



Il record di attivazione

```
void a();
void b();
void c();
int main(){
    ...
    a();
    ...
}
void a(){
    printf("Esecuzione di a\n");
    b();
    printf("Termine di a\n");
}
void b(){
    printf("Esecuzione di b\n");
    c();
    printf("Termine di b\n");
}
void c(){
    printf("Esecuzione di c\n");
    ...
    printf("Termine di c\n");
}
```

```
Esecuzione di a
Esecuzione di b
```

TERMINALE

Record di b

Record di a

Record di main

MEMORIA



Il record di attivazione

```
void a();
void b();
void c();
int main(){
    ...
    a();
    ...
}
void a(){
    printf("Esecuzione di a\n");
    b();
    printf("Termine di a\n");
}
void b(){
    printf("Esecuzione di b\n");
    c();
    printf("Termine di b\n");
}
void c(){
    printf("Esecuzione di c\n");
    ...
    printf("Termine di c\n");
}
```

```
Esecuzione di a
Esecuzione di b
Esecuzione di c
```

TERMINALE

Record di c
Record di b
Record di a
Record di main

MEMORIA



Il record di attivazione

```
void a();
void b();
void c();
int main(){
    ...
    a();
    ...
}
void a(){
    printf("Esecuzione di a\n");
    b();
    printf("Termine di a\n");
}
void b(){
    printf("Esecuzione di b\n");
    c();
    printf("Termine di b\n");
}
void c(){
    printf("Esecuzione di c\n");
    ...
    printf("Termine di c\n");
}
```

```
Esecuzione di a
Esecuzione di b
Esecuzione di c
Termine di c
```

TERMINALE

Record di c
Record di b
Record di a
Record di main

MEMORIA



Il record di attivazione

```
void a();
void b();
void c();
int main(){
    ...
    a();
    ...
}
void a(){
    printf("Esecuzione di a\n");
    b();
    printf("Termine di a\n");
}
void b(){
    printf("Esecuzione di b\n");
    c();
    printf("Termine di b\n");
}
void c(){
    printf("Esecuzione di c\n");
    ...
    printf("Termine di c\n");
}
```

```
Esecuzione di a
Esecuzione di b
Esecuzione di c
Termine di c
Termine di b
```

TERMINALE

Record di b
Record di a
Record di main

MEMORIA

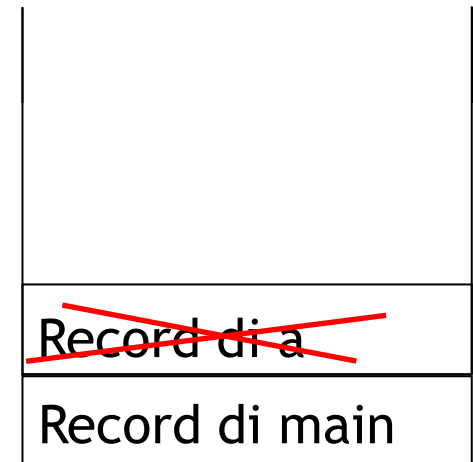


Il record di attivazione

```
void a();
void b();
void c();
int main(){
    ...
    a();
    ...
}
void a(){
    printf("Esecuzione di a\n");
    b();
    printf("Termine di a\n");
}
void b(){
    printf("Esecuzione di b\n");
    c();
    printf("Termine di b\n");
}
void c(){
    printf("Esecuzione di c\n");
    ...
    printf("Termine di c\n");
}
```

```
Esecuzione di a
Esecuzione di b
Esecuzione di c
Termine di c
Termine di b
Termine di a
```

TERMINALE



MEMORIA



Il record di attivazione

```
void a();
void b();
void c();
int main(){
    ...
    a();
    ...
}
void a(){
    printf("Esecuzione di a\n");
    b();
    printf("Termine di a\n");
}
void b(){
    printf("Esecuzione di b\n");
    c();
    printf("Termine di b\n");
}
void c(){
    printf("Esecuzione di c\n");
    ...
    printf("Termine di c\n");
}
```

```
Esecuzione di a
Esecuzione di b
Esecuzione di c
Termine di c
Termine di b
Termine di a
```

TERMINALE

Record di main

MEMORIA



Esecuzione di una funzione

Quando una funzione viene invocata, viene creato il suo record di attivazione in cui vengono memorizzate tutte le variabili usate nella funzione **inclusi i parametri formali**.

**Record di
attivazione del
chiamante**



**Record di
attivazione del
chiamato**



Esecuzione di una funzione

Quando una funzione viene invocata, viene creato il suo record di attivazione in cui vengono memorizzate tutte le variabili usate nella funzione **inclusi i parametri formali**.

- Da una funzione non ci può accedere alle variabili dichiarate dal chiamante o da altre funzioni (i record di attivazione sono isolati)

**Record di
attivazione del
chiamante**



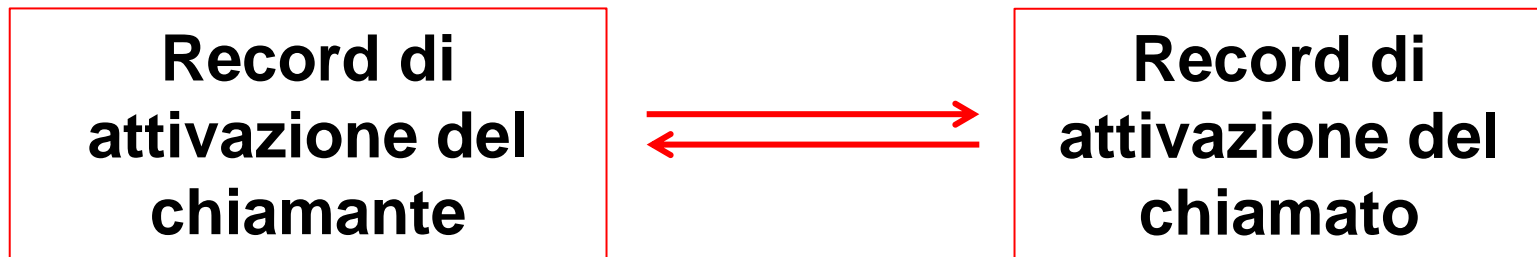
**Record di
attivazione del
chiamato**



Esecuzione di una funzione

Quando una funzione viene invocata, viene creato il suo record di attivazione in cui vengono memorizzate tutte le variabili usate nella funzione **inclusi i parametri formali**.

- Da una funzione non ci può accedere alle variabili dichiarate dal chiamante o da altre funzioni (i record di attivazione sono isolati)



Le comunicazioni tra i record di attivazione avvengono solamente mediante copia dei valori dei parametri in ingresso e con la return



Riepilogando: Esecuzione di una funzione

Quando viene invocata una funzione:

1. Vengono **calcolati** i valori dei **parametri attuali** di ingresso



Riepilogando: Esecuzione di una funzione

Quando viene invocata una funzione:

1. Vengono **calcolati** i valori dei **parametri attuali** di ingresso
2. Viene **creato il record di attivazione** per la funzione



Riepilogando: Esecuzione di una funzione

Quando viene invocata una funzione:

1. Vengono **calcolati** i valori dei **parametri attuali** di ingresso
2. Viene **creato il record di attivazione** per la funzione
3. I **valori** dei **parametri attuali** di ingresso vengono **copiati** nei **parametri formali** all'interno del **record di attivazione della funzione**
 - Il record di attivazione della funzione ora contiene solamente i parametri formali con assegnati i valori dei parametri attuali



Riepilogando: Esecuzione di una funzione

Quando viene invocata una funzione:

1. Vengono **calcolati** i valori dei **parametri attuali** di ingresso
2. Viene **creato il record di attivazione** per la funzione
3. I **valori** dei **parametri attuali** di ingresso vengono **copiati** nei **parametri formali** all'interno del **record di attivazione della funzione**
 - Il record di attivazione della funzione ora contiene solamente i parametri formali con assegnati i valori dei parametri attuali
4. Viene **eseguito il corpo** della **funzione** che eventualmente dichiarerà altre variabili



Riepilogando: Esecuzione di una funzione

Quando viene invocata una funzione:

1. Vengono **calcolati** i valori dei **parametri attuali** di ingresso
2. Viene **creato il record di attivazione** per la funzione
3. I **valori** dei **parametri attuali** di ingresso vengono **copiati** nei **parametri formali** all'interno del **record di attivazione della funzione**
 - Il record di attivazione della funzione ora contiene solamente i parametri formali con assegnati i valori dei parametri attuali
4. Viene **eseguito il corpo** della **funzione** che eventualmente dichiarerà altre variabili
5. Viene **copiato il valore di ritorno**



Riepilogando: Esecuzione di una funzione

Quando viene invocata una funzione:

1. Vengono **calcolati** i valori dei **parametri attuali** di ingresso
2. Viene **creato il record di attivazione** per la funzione
3. I **valori** dei **parametri attuali** di ingresso vengono **copiati** nei **parametri formali** all'interno del **record di attivazione della funzione**
 - Il record di attivazione della funzione ora contiene solamente i parametri formali con assegnati i valori dei parametri attuali
4. Viene **eseguito il corpo** della **funzione** che eventualmente dichiarerà altre variabili
5. Viene **copiato il valore di ritorno**
6. Il record di attivazione della funzione viene **distrutto**



Esecuzione di una funzione: esempio

```
int x, w, r;
```

```
(1) x=3;
```

```
(2) w=2;
```

```
(3) r = funz(4);
```

```
int funz(x) {  
    int y, z;  
    y = 2*x;      %(1')  
    x = 0;        %(2')  
    z = 4;        %(3')  
    return y;  
}
```

RA chiamante

```
x = ???
```

```
w = ???
```

```
r = ???
```



Esecuzione di una funzione: esempio

```
int x, w, r;
```

```
(1) x=3;
```

```
(2) w=2;
```

```
(3) r = funz(4);
```

```
int funz(x) {  
    int y, z;  
    y = 2*x;      %(1')  
    x = 0;        %(2')  
    z = 4;        %(3')  
    return y;  
}
```

RA chiamante dopo (1)

```
x=3
```

```
w = ???
```

```
r = ???
```



Esecuzione di una funzione: esempio

```
int x, w, r;
```

```
(1) x=3;
```

```
(2) w=2;
```

```
(3) r = funz(4);
```

```
int funz(x) {  
    int y, z;  
    y = 2*x;      %(1')  
    x = 0;        %(2')  
    z = 4;        %(3')  
    return y;  
}
```

RA chiamante dopo (2)

```
x=3
```

```
w = 2
```

```
r = ???
```



Esecuzione di una funzione: esempio

```
int x, w, r;
```

```
(1) x=3;
```

```
(2) w=2;
```

```
(3) r = funz(4);
```

```
int funz(x) {  
    int y, z;  
    y = 2*x;      %(1')  
    x = 0;        %(2')  
    z = 4;        %(3')  
    return y;  
}
```

RA chiamante durante (3)

```
x=3  
w = 2  
r = ???
```

RA chiamato prima (1')

```
x=4  
y = ???  
z = ???
```



Esecuzione di una funzione: esempio

```
int x, w, r;
```

```
(1) x=3;
```

```
(2) w=2;
```

```
(3) r = funz(4);
```

```
int funz(x) {  
    int y, z;  
    y = 2*x;      %(1')  
    x = 0;        %(2')  
    z = 4;        %(3')  
    return y;  
}
```

RA chiamante durante (3)

```
x=3  
w = 2  
r = ???
```

RA chiamato dopo (1')

```
x=4  
y = 8  
z = ???
```



Esecuzione di una funzione: esempio

```
int x, w, r;
```

```
(1) x=3;
```

```
(2) w=2;
```

```
(3) r = funz(4);
```

```
int funz(x) {  
    int y, z;  
    y = 2*x;      %(1')  
    x = 0;        %(2')  
    z = 4;        %(3')  
    return y;  
}
```

RA chiamante durante (3)

```
x=3  
w = 2  
r = ???
```

RA chiamato dopo (2')

```
x=0  
y = 8  
z = ???
```



Esecuzione di una funzione: esempio

```
int x, w, r;
```

```
(1) x=3;
```

```
(2) w=2;
```

```
(3) r = funz(4);
```

```
int funz(x) {  
    int y, z;  
    y = 2*x;      %(1')  
    x = 0;        %(2')  
    z = 4;        %(3')  
    return y;  
}
```

RA chiamante durante (3)

```
x=3  
w = 2  
r = ???
```

RA chiamato dopo (3')

```
x=0  
y=8  
z=4
```



Esecuzione di una funzione: esempio

```
int x, w, r;
```

```
(1) x=3;
```

```
(2) w=2;
```

```
(3) r = funz(4);
```

```
int funz(x) {  
    int y, z;  
    y = 2*x;      %(1')  
    x = 0;        %(2')  
    z = 4;        %(3')  
    return y;  
}
```

RA chiamante dopo (3)

```
x=3
```

```
w = 2
```

```
r = 8
```




La struttura di un programma

Parte direttiva

- `#include`
- `#define`

Parte dichiarativa globale

- Dichiarazioni di tipi
- Prototipi dei sottoprogrammi

Programma principale (`main`)

- Parte dichiarativa (variabili e costanti locali)
- Parte esecutiva

Definizione dei sottoprogrammi

- Parte dichiarativa (variabili e costanti locali)
- Parte esecutiva