

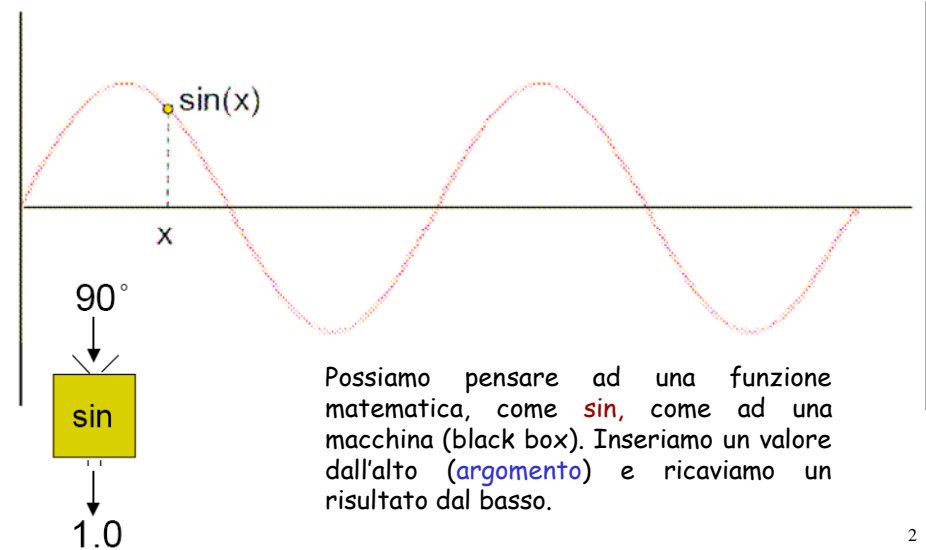


Insegnamento di Programmazione (6 CFU)

Funzioni

Ing. Nadia Ranaldo
Dipartimento di Ingegneria
Università degli Studi del Sannio

Funzioni dalla matematica (1)

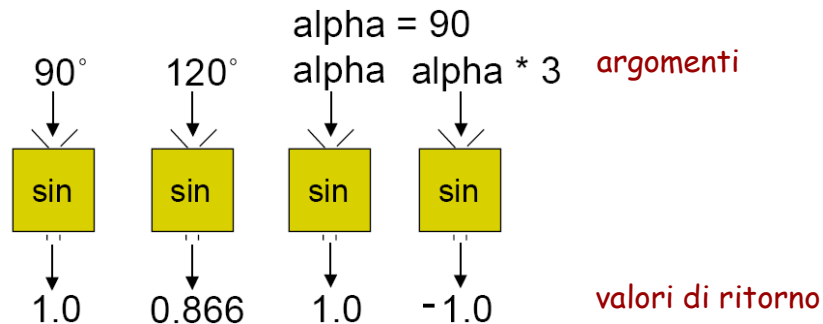


2

Funzioni dalla matematica (2)



- Cambiando l'argomento, la funzione ritorna un risultato differente. L'argomento può essere specificato come una costante, una variabile o un'espressione
 - I risultati possono essere utilizzati per fare altre operazioni
- `total = sin(90)+sin(120)+sin(alpha)+sin(alpha*3)`



3

Funzioni del C (1)



- Il C fornisce la funzione `sin` in una libreria standard chiamata `math.h`. L'argomento deve essere in radianti.

```
#include <math.h>
main() {
    double alpha, result;
    alpha = 90;
    result = sin(alpha * 3.14 / 180);
    printf("sin of %g is %g", alpha, result);
}
```

%f: formato con punto decimale

%e: formato in notazione scientifica ($1.2e2 = 1.2 \cdot 10^2$)

%g: formato generale: usa `%f` o `%e` a seconda di quale risulta più breve

4

Funzioni del C (2)



- Usando #define il codice è più semplice da leggere.
- Se una define utilizza un'espressione, dovrebbe sempre includere le parentesi, per evitare la possibilità che la precedenza tra gli operatori possa alterarne il significato

```
#include <stdio.h>
#include <math.h>
#define PI 3.14
#define RADIAN_FACTOR (PI / 180)
main() {
    double alpha, result;
    alpha = 90;
    result = sin(alpha * RADIAN_FACTOR);
    printf("sin of %g is %g", alpha, result);
}
```

5

Funzioni del C (3)



- Un argomento di una funzione può essere il valore ritornato da un'altra.

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159
#define RADIAN_FACTOR (PI / 180)
main() {
    double alpha, result;
    alpha = 90;
    printf("sin of %g is %g", alpha, sin(alpha *
    RADIAN_FACTOR));
}
```

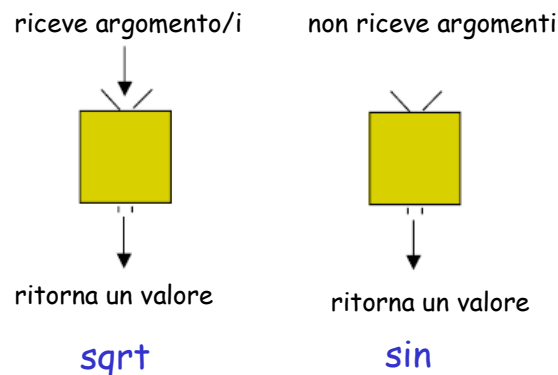
Chiamate a funzioni (function call)

6

Tipi di funzioni (1)



- Funzioni che calcolano e restituiscono un valore di ritorno

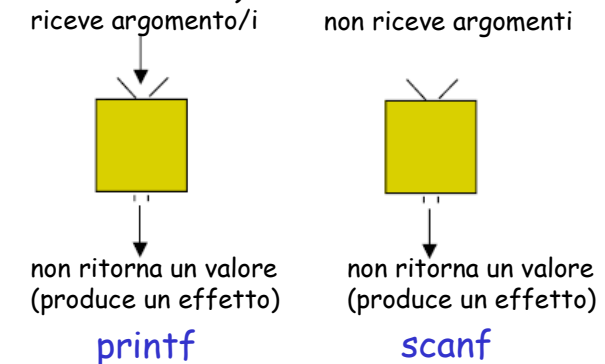


7

Tipi di funzioni (2)



- Funzioni che non restituiscono un valore di ritorno: vengono eseguite per il loro effetto
- In alcuni linguaggio sono chiamate procedure (in C funzioni e procedure sono trattate allo stesso modo, si utilizza la stessa sintassi)



8

Scrivere le proprie funzioni



- Scrivere una nuova funzione richiede:
 - Un prototipo
 - Il codice che implementa la funzione
 - Una o più chiamate della funzione

```
prototipo
main() {
    . . .
    chiamata della funzione
    . . .
}

implementazione
```

9

Chiamata della funzione



- Esempio: un programma che usa una semplice funzione che ritorna un valore pari al doppio del suo argomento

```
main() {
    int number;
    printf("Inserire un numero: ");
    scanf("%d",&number);
    printf("Il doppio del numero inserito
    e'%d", doppio(number));
}
```

10

Implementazione



```
int doppio(int x) {
    return (2*x);
}
```

Diagram annotations:

- tipo di ritorno (red arrow pointing to `int`)
- nome della funzione (black arrow pointing to `doppio`)
- parametro formale (green arrow pointing to `int x`)
- tipo (green arrow pointing to `int`)
- nome (green arrow pointing to `x`)
- istruzione di ritorno (blue arrow pointing to `return (2*x);`)
- valore di ritorno (black arrow pointing to `2*x`)

- Sintassi dell'istruzione di ritorno:
`return (expression);`
- Se la funzione non restituisce nulla la sintassi è `return;` e può essere omesso

11

Prototipo



- Il prototipo della funzione deve essere specificato prima dell'implementazione delle funzioni che la utilizzano (in questo caso prima del main)

```
#include<stdio.h>
int doppio(int x);
main() {
    int number;
    printf("Inserire un numero: ");
    scanf("%d",&number);
    printf("Il doppio del numero inserito
    e'%d", doppio(number));
}
```

12

Prototipo: sintassi



- Corrisponde alla dichiarazione di una funzione (in maniera analoga alla dichiarazione di una variabile)

```
result-type name (parameters);
```

Dove **result-type** è il tipo del valore ritornato dalla funzione, **parameters** è una lista di **parametri formali** (o brevemente parametro) separati dalla virgola. Un parametro è specificato da un tipo e, in maniera opzionale, da un nome per dare maggiori informazioni al programmatore che vuole usare la funzione

Esempi

```
double sqrt(double);  
double sin (double angleInRadians);
```

13

Esempio completo



```
#include<stdio.h>  
int doppio(int x);  
main() {  
    int number;  
    printf("Inserire un numero: ");  
    scanf("%d",&number);  
    printf("Il doppio del numero inserito e' %d",  
        doppio(number));  
}  
int doppio(int value) {  
    return (2*value);  
}
```

Il nome dei parametri usato per l'implementazione della funzione può essere diverso da quello usato nel prototipo e non influenza il risultato della funzione!

14

Esempio. Il fattoriale



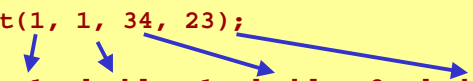
```
#include<stdio.h>  
int fattoriale(int n);  
main() {  
    int number;  
    printf("Inserire un numero: ");  
    scanf("%d",&number);  
    printf("Il fattoriale del numero inserito  
e' %d", fattoriale(number));  
}  
int fattoriale(int n) {  
    int i, fatt=1;  
    for(i=n;i>=1;i--)  
        fatt*=i;  
    return fatt;  
}
```

15

Funzioni con più parametri (1)



```
double dist(double x1, double y1, double x2, double y2);  
main() {  
    double separation;  
    separation = dist(1, 1, 34, 23);  
}  
double dist(double x1, double y1, double x2, double y2) {  
    double d;  
    d = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));  
    return (d);  
}
```



- Corrispondenza uno-a-uno degli argomenti con i parametri formali in base alla posizione
- I tipi sono convertiti automaticamente se necessario

16

Funzioni con più parametri (2)



```
main() {
    double separation, x1, y1;
    int i = 2;
    x1 = 12.0;
    y1 = 25.0;
    separation = dist(1, i, x1, sqrt(y1) + 2);
}
double dist(double x1, double y1, double x2, double y2){
    double d;
    d = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    return (d);
}
```

- Corrispondenza uno-a-uno degli argomenti con i parametri formali in base alla posizione e **non al nome!**

Funzioni senza parametri e senza valore di ritorno



```
void sayHi(void);
main(){
    sayHi();
}
void sayHi(void) {
    printf("Hi\n");
}
```

nessun valore di ritorno

nessun parametro (può essere omesso)

istruzione di ritorno non necessaria (return;)

- Funzione invocata per produrre un effetto, ovvero la stampa a video di Hi

Funzioni predicato (1)



valore di ritorno intero

```
#define TRUE 1
#define FALSE 0
int isEven(int n);
main(){
    int i=...
    if(isEven(i)==TRUE)
        ...
    else ...
}
int isEven(int x) {
    if(x%2!=0)
        return FALSE;
    else return TRUE;
}
```

- Funzioni che restituiscono un valore logico vero o falso (**TRUE** o **FALSE**)
- Usate spesso per prendere delle decisioni nelle strutture di controllo

Funzioni predicato (2)



Scrittura sintetica:
se il valore restituito dalla funzione è !=0 allora la condizione è vera altrimenti è falsa

```
#define TRUE 1
#define FALSE 0
int isEven(int n);
main(){
    int i=...
    if(isEven(i))
        ...
    else ...
}
int isEven(int x) {
    if(x%2)
        return FALSE;
    else return TRUE;
}
```

Esempio di funzione



- Scrivere una funzione `RaiseIntToPower` che prende due interi `n` e `k` e restituisce n^k .
- Algoritmo: calcolare $n*n*...*n$ ovvero effettuare per `k-1` volte la moltiplicazione tra `n` e il risultato parziale calcolato al passo precedente

```
long raiseIntToPower(int n,int k) {
    int i;
    long val=1;
    for (i=0;i<k;i++) {
        val*=n;
    }
    return val;
}
```

21

Chiamate a funzioni innestate (1)



- Una volta definita una funzione, questa può essere utilizzata non soltanto in un programma principale (`main`) ma anche **come strumento per implementare funzioni più complesse**.
- Esempio: calcolare quante sono le possibili combinazioni `C` su `k` posti di un insieme di `n` oggetti distinti
- Ad esempio `n=3` oggetti (1,2,3) `k=2` $C(n,k)?$
- Le combinazioni su due posti sono (1,2) (2,3) e (1,3) quindi $C = 3$

`C` ha una definizione semplice in termine di fattoriali:

$$C(n,k) = \frac{n!}{k! \times (n-k)!}$$

Ad esempio:

$$C(3,2) = \frac{3!}{2! \times (3-2)!} = \frac{6}{2 \times 1} = 3$$

22

Chiamate a funzioni innestate (2)



- Utilizzando la funzione fattoriale (già implementata) l'implementazione della funzione combinazioni richiede solo la traduzione della sua definizione matematica:

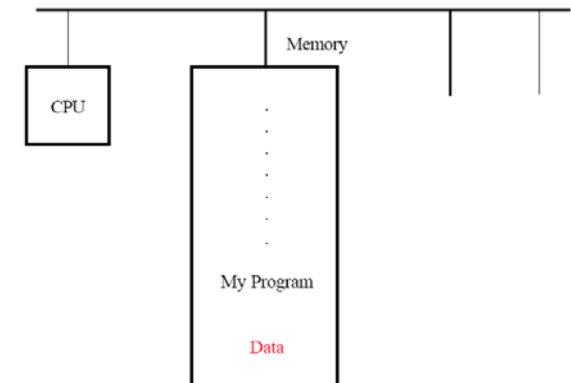
```
int combinazioni (int n, int k) {
    return
    (fattoriale(n)/(fattoriale(k)*(fattoriale(n-k)));
}
/* un semplice programma che usa la funzione*/
main(){
    int n,k;
    n=3;
    k=2;
    printf("C(%d,%d)=%d\n",n,k,combinazioni(n,k));
}
```

23

Meccanismo di chiamata a funzione



- Per comprendere il funzionamento delle chiamate a funzioni, occorre comprendere cosa accade nella memoria del computer quando viene eseguita una funzione
- Quando viene eseguito un programma, una parte della memoria è riservata a contenere il **codice** ed un'altra i **dati**

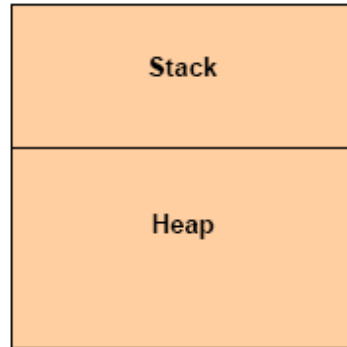


24

Meccanismo di chiamata a funzione: la memoria dati (1)



- La parte riservata ai dati si compone a sua volta di due parti:
 - Lo stack: usato per contenere i dati statici
 - L'heap: usato per contenere i dati dinamici

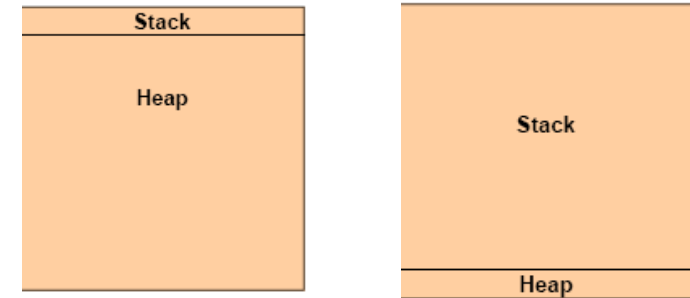


25

Meccanismo di chiamata a funzione: la memoria dati (2)



- La parte di stack non utilizzata viene utilizzata dall'heap
- Lo stack può avere una dimensione massima, fissata dall'utente
- E' possibile che si verifichi un errore quando si supera tale dimensione (stack overflow error)



26

Meccanismo di chiamata a funzione: la memoria dati (3)



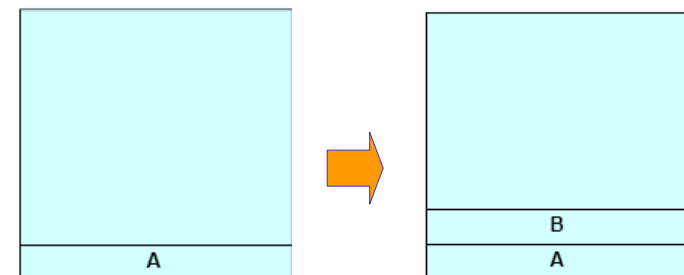
- Dati statici
 - Variabili dichiarate esplicitamente all'inizio delle funzioni
 - Hanno un nome
 - Necessarie all'esecuzione del programma
- Dati dinamici
 - Variabili dichiarati durante l'esecuzione del programma
 - Non hanno un nome
 - Non sono necessari, ma permettono di scrivere programmi potenti

27

Meccanismo di chiamata a funzione: lo stack (1)



- Il concetto di stack: un modo di gestire le cose
 - Operazioni di **push** (inserire al top dello stack) e di **pop** (prelevare dal top dello stack)
 - E' possibile prelevare un dato solo al top dello stack, ovvero l'ultimo elemento che è stato inserito
- Esempio push(A), poi push (B)

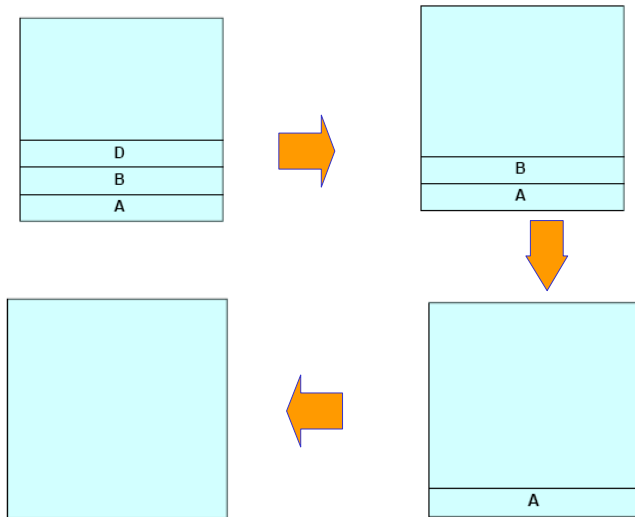


28

Meccanismo di chiamata a funzione: lo stack (2)



Esempio push(D), poi pop, pop, pop

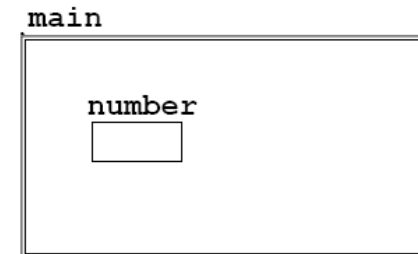


29

Meccanismo di chiamata a funzione: lo stack frame (1)



- All'inizio dell'esecuzione della funzione main, viene creato nella parte della memoria chiamata stack uno stack frame per la funzione main
 - una porzione di memoria riservata a contenere le variabili locali della funzione, ovvero le variabili dichiarate all'interno della funzione e le variabili associate ai parametri formali
- Ogni chiamata di funzione porta alla creazione di un nuovo stack frame al top dello stack contenente variabili locali separate



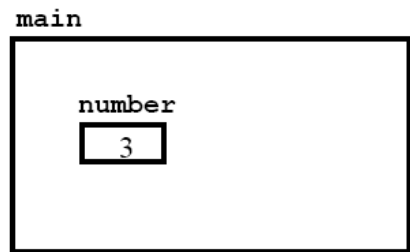
30

Meccanismo di chiamata a funzione: lo stack frame (2)



```
main() {
    int number;
    printf("Inserire un numero: ");
    scanf("%d",&number);
    printf("Il doppio del numero inserito e' %d",
        doppio(number));
}
```

- Dopo l'esecuzione della scanf la variabile locale number nello stack frame del main contiene il valore inserito dall'utente



31

Meccanismo di chiamata a funzione: lo stack frame (3)

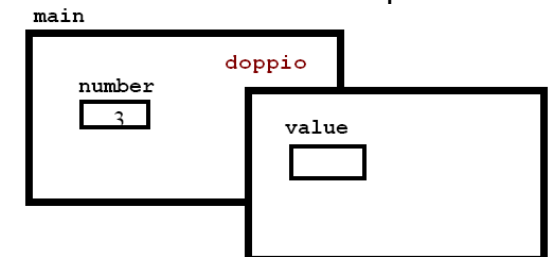


```
main() {
    int number;
    printf("Inserire un numero: ");
    scanf("%d",&number);
    printf("Il doppio del numero inserito e' %d",
        doppio(number));
}
```

- Per eseguire la printf, occorre valutare i suoi argomenti.
- Per prima cosa viene invocata la funzione doppio e quindi viene creato uno stack frame (al top dello stack) per tale funzione contenente la variabile value corrispondente al parametro formale.

```
int doppio(int value){
    return (2*value);
}
```

Le variabili sono locali alla funzione in cui sono dichiarate!



32

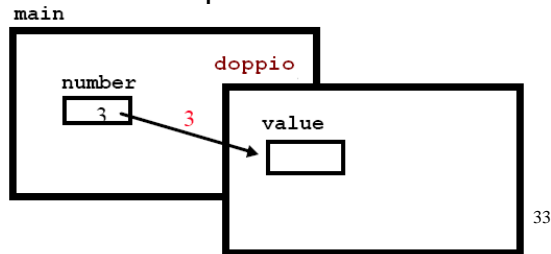


Meccanismo di chiamata a funzione: passaggio dei parametri (1)



```
main() {
  int number;
  printf("Inserire un numero: ");
  scanf("%d",&number);
  printf("Il doppio del numero inserito e' %d",
        doppio(number));
}
int doppio(int value){
  return (2*value);
}
```

- Quando la funzione main invoca la funzione doppio, il valore corrente dell'argomento (in questo caso number) viene **passato** e **copiato** nella corrispondente variabile del parametro formale.



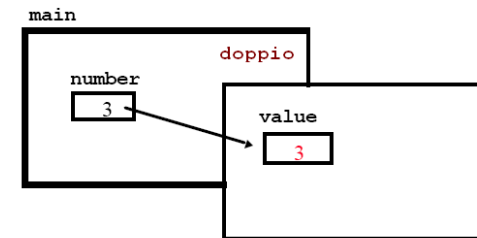
33



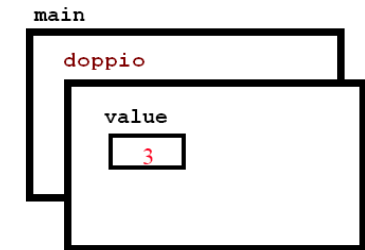
Meccanismo di chiamata a funzione: passaggio dei parametri (2)



- La variabile value contiene il valore della variabile number



- Le variabili nella funzione chiamante (main) non sono accessibili durante l'esecuzione della funzione doppio



34

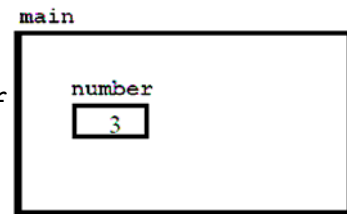


Meccanismo di chiamata a funzione: valore di ritorno



- Il valore di ritorno è utilizzato nell'espressione in cui si trova la chiamata a funzione.
- Al termine dell'esecuzione della funzione:
 - il relativo stack frame viene distrutto (pop), ovvero la memoria occupata viene liberata e può essere utilizzata per creare altri stack frame
 - l'esecuzione riprende dal punto in cui era stata interrotta (indirizzo di ritorno)
- In questo esempio viene creato successivamente lo stack frame per la printf che viene distrutto dopo l'esecuzione

```
main() {
  int number;
  printf("Inserire un numero: ");
  scanf("%d",&number);
  printf("Il doppio del numero inserito e' %d",
        doppio(number));
}
int doppio(int value){
  return (2*value);
}
```



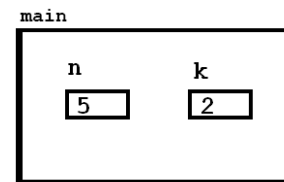
35



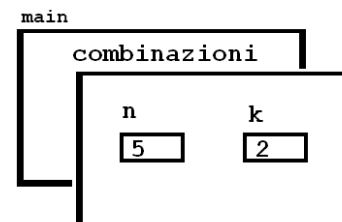
Meccanismo di chiamata a funzione: esempio (1)



- Cosa avviene nello stack durante l'esecuzione del programma che utilizza la funzione combinazioni?



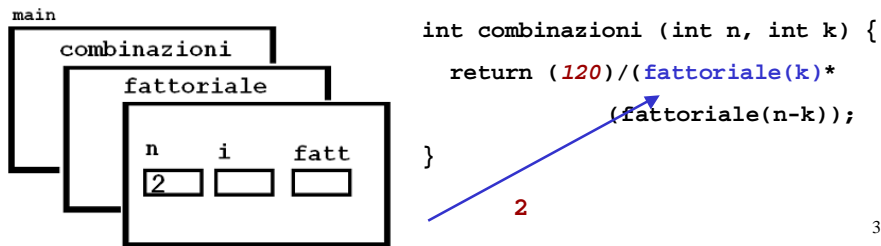
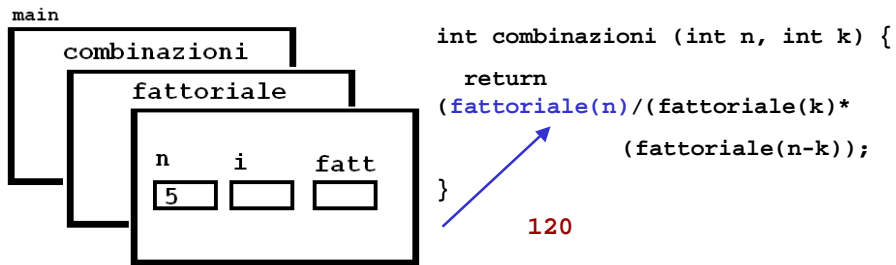
```
main(){
  int n,k;
  n=5;
  k=2;
  printf("C(%d,%d)=%d\n",n,k,
        combinazioni(n,k));
}
```



```
main(){
  int n,k;
  n=3;
  k=2;
  printf("C(%d,%d)=%d\n",n,k,
        combinazioni(n,k));
}
```

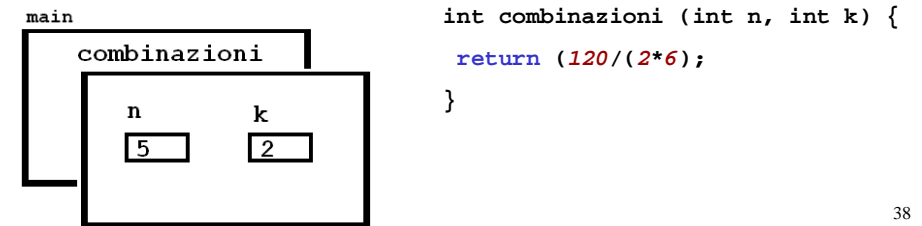
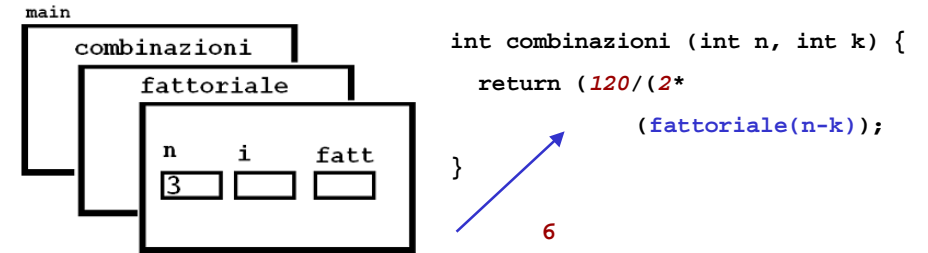
36

Meccanismo di chiamata a funzione: esempio (2)



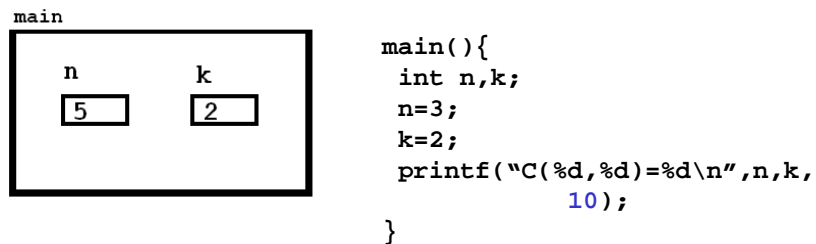
37

Meccanismo di chiamata a funzione: esempio (3)



38

Meccanismo di chiamata a funzione: esempio (4)



39