# Programmare in C – Primi passi

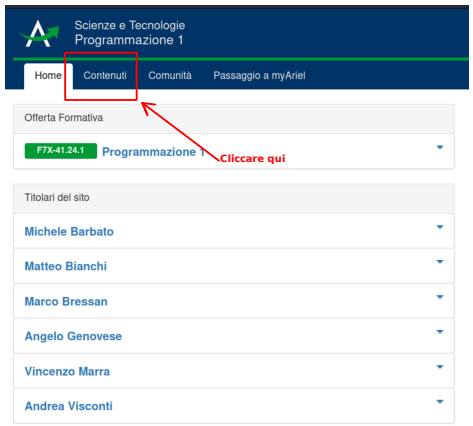
**Michele Barbato**

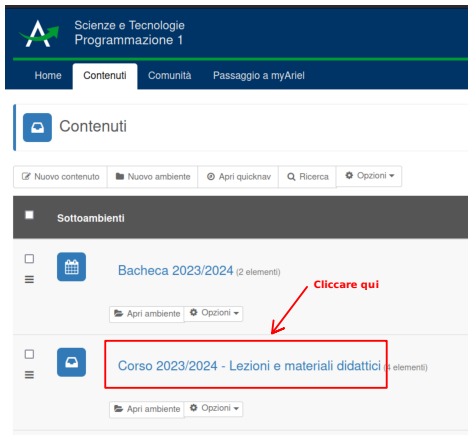
Dipartimento di Informatica *“Giovanni degli Antoni”* – Università degli Studi di Milano

UNIVERSITÀ
DEGLI STUDI
DI MILANO

# Scaricare gli Esercizi

- file PDF con gli **esercizi** su **ariel.unimi.it**
  - Fare il **login**
  - Accedere alla pagina “**Programmazione 1**”
  - Nella sezione “Contenuti” selezionare “**Corso 2023/2024 - Lezioni e materiali didattici**”
  - Cliccare su “**Vedi file allegati**” dell’argomento “**Laboratorio di Programmazione - Lezione 1**”
  - **Scaricare** il file allegato (**lab<n>.pdf**)



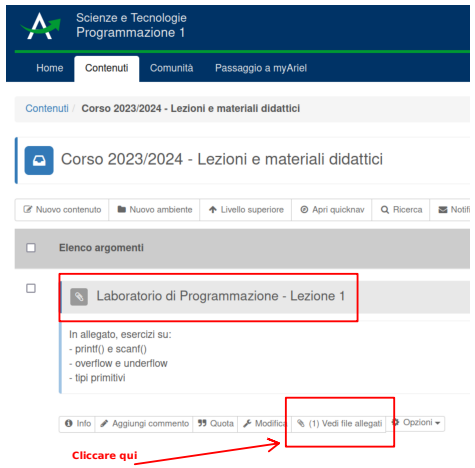




Figura: Passi per scaricare il file PDF con gli esercizi da svolgere in laboratorio

# Riassunto

## Organizzazione

- Gruppo (intervallo cognomi): A-Dig. **Rispettare questa suddivisione**
- Materiale didattico generale: su Ariel
- Slide laboratorio su `upload.di.unimi.it`
  (**Accedere alla sezione corretta**)

## BASH

- `cd` cambia directory;
- `touch` e `mkdir` creano file e cartelle;
- `pluma` lancia l'editor di testo;
- `gcc` compila

# Suggerimenti

- 1 file sorgente ( `<nome_file>.c` ) per ogni esercizio (**non saltateli**)
- file sorgente in `Documents/prog_23_24/lab1` (creare `lab1` )
- se in difficoltà con BASH → **OK interfaccia grafica** (mouse, cartelle, ecc.)
- **compilare** dal **terminale** ed **eseguire**:
  1. `gcc <nome_sorgente.c> -o <nome_eseguibile>`
  2. `./<nome_eseguibile>`

**Esempio:**

- il sorgente si chiama `esercizio1.c` e voglio chiamare l'eseguibile `ese1` :
  1. `gcc esercizio1.c -o ese1`
  2. `./ese1`

# Suggerimenti

- 1 file sorgente ( `<nome_file>.c` ) per ogni esercizio (**non saltateli**)
- file sorgente in `Documents/prog_23_24/lab1` (creare `lab1` )
- se in difficoltà con BASH → **OK interfaccia grafica** (mouse, cartelle, ecc.)
- **compilare** dal **terminale** ed **eseguire**:
  1. `gcc <nome_sorgente.c> -o <nome_eseguibile>`
  2. `./<nome_eseguibile>`

**Esempio:**

- il sorgente si chiama `esercizio1.c` e voglio chiamare l'eseguibile `ese1` :
  1. `gcc esercizio1.c -o ese1`
  2. `./ese1`

**Esercitarsi a casa**

- se non avete GNU/Linux, esistono compilatori online: `https://www.onlinegdb.com/online_c_compiler`
- **IMPORTANTE:** a **lezione** e all'**esame**, il vostro **codice DEVE compilare ed eseguire** correttamente sulle **macchine del laboratorio**

# Le buone pratiche

## Nel sorgente...

- ...date **nomi sensati** alle **variabili**
- ...**indentate**: le **righe** di uno **stesso blocco** sono **allineate** verticalmente e **più interne** rispetto ai **blocchi superiori**
- ...**commenti stringati** e solo se necessari (e.g., se fate cose complicate)

## Esempio:

### NO

```
#include<stdio.h>
#include<stdbool.h>
        int main(void)
        {
        //a e' "false" se non e' la 1a lezione
        bool a = false ;
        if (!a)
        {
        printf("Vi prego, indentate\n") ;
        }
        else
        {
        printf("OK") ;
}
        return 0 ;
}
```

### SÌ

```
#include<stdio.h>
#include<stdbool.h>
int main()
{
        bool prima_lezione = true ;
        if (prima_lezione)
        {
                printf("Vi prego, indentate\n") ;
        }
        else
        {
                printf("OK") ;
        }

        return 0 ;
}
```

# Note su `printf`

- usiamo `printf` per **mostrare testo**
- un testo è una **sequenza di caratteri**
- nel codice usiamo "**simboli**" per **rappresentare i caratteri**
  - quasi sempre i caratteri e i simboli coincidono:
    simbolo `A` per il carattere `A`, simbolo `b` per il carattere `b`,...
  - a volte, occorre usare simboli speciali (con `\` (backslash)):
    simbolo `\'` per il carattere `'`, simbolo `\n` per il carattere "a capo"

# Note su `printf`

- usiamo `printf` per **mostrare testo**
- un testo è una **sequenza di caratteri**
- nel codice usiamo "**simboli**" per **rappresentare i caratteri**
  - quasi sempre i caratteri e i simboli coincidono:
    simbolo `A` per il carattere `A`, simbolo `b` per il carattere `b`,...
  - a volte, occorre usare simboli speciali (con `\` (backslash)):
    simbolo `\'` per il carattere `'`, simbolo `\n` per il carattere "a capo"
- **"internamente"** però i **caratteri** sono numeri **binari** (sequenze di bit)
  - vedere codice ASCII, colonna "Binary"
- di conseguenza, i **caratteri** corrispondono a **valori interi**
  - `printf("%c", 65)` mostra `A` (65 interpretato come carattere)
  - `printf("%d", 'A')` mostra `65` (`'A'` interpretato come intero)

# Note su `printf`

- usiamo `printf` per **mostrare testo**
- un testo è una **sequenza di caratteri**
- nel codice usiamo "**simboli**" per **rappresentare i caratteri**
  - quasi sempre i caratteri e i simboli coincidono:
    simbolo `A` per il carattere `A`, simbolo `b` per il carattere `b`,...
  - a volte, occorre usare simboli speciali (con `\` (backslash)):
    simbolo `\'` per il carattere `'`, simbolo `\n` per il carattere "a capo"
- **"internamente"** però i **caratteri** sono numeri **binari** (sequenze di bit)
  - vedere codice ASCII, colonna "Binary"
- di conseguenza, i **caratteri** corrispondono a **valori interi**
  - `printf("%c", 65)` mostra `A` (65 interpretato come carattere)
  - `printf("%d", 'A')` mostra `65` (`'A'` interpretato come intero)
- valori esprimibili anche in **base 8** (notazione `\<numero>`) o in **base 16** (notazione `\x<numero>`)
  - `printf("\101")` mostra `A` (usando la base 8)
  - `printf("%o", 'A')` mostra `101` (65 in base 8)

# Note su `printf`

- usiamo `printf` per **mostrare testo**
- un testo è una **sequenza di caratteri**
- nel codice usiamo "**simboli**" per **rappresentare i caratteri**
  - quasi sempre i caratteri e i simboli coincidono:
    simbolo `A` per il carattere `A`, simbolo `b` per il carattere `b`,...
  - a volte, occorre usare simboli speciali (con `\` (backslash)):
    simbolo `\'` per il carattere `'`, simbolo `\n` per il carattere "a capo"
- **"internamente"** però i **caratteri** sono numeri **binari** (sequenze di bit)
  - vedere codice ASCII, colonna "Binary"
- di conseguenza, i **caratteri** corrispondono a **valori interi**
  - `printf("%c", 65)` mostra `A` (65 interpretato come carattere)
  - `printf("%d", 'A')` mostra `65` (`'A'` interpretato come intero)
- valori esprimibili anche in **base 8** (notazione `\<numero>`) o in **base 16** (notazione `\x<numero>`)
  - `printf("\101")` mostra `A` (usando la base 8)
  - `printf("%o", 'A')` mostra `101` (65 in base 8)
  - `printf("\x41")` mostra `A` (usando la base 16)
  - `printf("%x", 'A')` mostra `41` (65 in base 16)

# Note su `printf`

- usiamo `printf` per **mostrare testo**
- un testo è una **sequenza di caratteri**
- nel codice usiamo "**simboli**" per **rappresentare i caratteri**
  - quasi sempre i caratteri e i simboli coincidono:
    simbolo `A` per il carattere `A`, simbolo `b` per il carattere `b`,...
  - a volte, occorre usare simboli speciali (con `\` (backslash)):
    simbolo `\'` per il carattere `'`, simbolo `\n` per il carattere "a capo"
- **"internamente"** però i **caratteri** sono numeri **binari** (sequenze di bit)
  - vedere codice ASCII, colonna "Binary"
- di conseguenza, i **caratteri** corrispondono a **valori interi**
  - `printf("%c", 65)` mostra `A` (65 interpretato come carattere)
  - `printf("%d", 'A')` mostra `65` (`'A'` interpretato come intero)
- valori esprimibili anche in **base 8** (notazione `\<numero>`) o in **base 16** (notazione `\x<numero>`)
  - `printf("\101")` mostra `A` (usando la base 8)
  - `printf("%o", 'A')` mostra `101` (65 in base 8)
  - `printf("\x41")` mostra `A` (usando la base 16)
  - `printf("%x", 'A')` mostra `41` (65 in base 16)

**Nota:** usare i simboli `\` e `%` solo tra coppie di doppi apici

# Un “trucco” per l’uso di `printf`

## Quando usate printf

- Se la vostra **stringa** di testo è **troppo lunga**, potete “**spezzarla**”
- **ogni “pezzo”** va in una **coppia di doppi apici** ( `"<testo>"` )
- gli **spazi** e i ritorni **a capo** tra i vari pezzi sono **ignorati** (cioè i pezzi vengono concatenati)

**Esempio:**

Le seguenti istruzioni sono equivalenti (output: `Ciao Mondo`):

```
/* 1. stringa unica */
printf("Ciao Mondo") ;

/* 2. stringa spezzata con spazio */
printf("Ciao" " Mondo") ;

/* 3. stringa spezzata con ritorno a capo */
printf("Ciao"
       " Mondo") ;

/* 4. doppio printf() */
printf("Ciao") ;
printf(" Mondo") ;
```

# Variabili

- Immaginate la **memoria** utilizzata dal programma come un **“magazzino a scaffale”** (`https://images.app.goo.gl/FptNZwhr6SwPVpmM9`)
- una **variabile** è una **“scatola”** del magazzino
- il **nome** della variabile è l’**identificativo** della scatola
- il **tipo** della variabile è la **tipologia di contenuto** ammesso nella scatola (per esempio, nell’immagine del link, identificata dal colore della scatola)
- il **valore** della variabile è il **contenuto** della scatola
- l’**indirizzo** della variabile è la **posizione** della scatola nella scaffalatura

**Esempio:**

`int x = 5 ;` può essere visto come una scatola:

- identificata dal nome `x`
- che contiene valori **interi**
- il cui **contenuto** attualmente è **5**

# Variabili: valore e indirizzo

Una volta dichiarata una variabile `x`

- il **nome** `x` ne identifica il **valore** (contenuto attuale della scatola)
- `&x` ne identifica l'**indirizzo** (posizione attuale della scatola)

**Esempio:**

`int x = 5 ;` può essere visto come una scatola:

- la cui **posizione** nel magazzino si indica con `&x`

# Variabili: valore e indirizzo

Una volta dichiarata una variabile `x`

- il **nome** `x` ne identifica il **valore** (contenuto attuale della scatola)
- `&x` ne identifica l'**indirizzo** (posizione attuale della scatola)

**Esempio:**
L'ultima delle tre istruzioni

- `int x = 5 ;`
- `int a = 3 ;`
- `x = a ;`

si può leggere come segue: "il **contenuto** della scatola `x` deve diventare **uguale** all'attuale **contenuto** della scatola `a`."

- Quindi, **dopo** la terza istruzione, **il valore di** `x` **è 3**.

# Variabili: valore e indirizzo

Una volta dichiarata una variabile `x`

- il **nome** `x` ne identifica il **valore** (contenuto attuale della scatola)
- `&x` ne identifica l'**indirizzo** (posizione attuale della scatola)

**Esempio:**
L'ultima delle due istruzioni

- `int x = 5 ;`
- `int x = x + 1 ;`

si può leggere come segue: "il **contenuto** della scatola `x` deve diventare **uguale** all'attuale **contenuto** della scatola `x` aumentato di 1."

- Quindi, **dopo** la terza istruzione, **il valore di** `x` **è 6**.

# La funzione `scanf()`

- `scanf()` **legge** un **input** inserito da un **utente**
- La struttura dell'**input non** è nel pieno **controllo** del programmatore
  - **Esempio:** l'utente usa una **virgola** per **separare due input** destinati alla stessa chiamata di `scanf()`
  - **Esempio:** l'utente inserisce un `float` dove andrebbe un `int`

- In seguito, **consigli generali**. Considerare **sempre**:
  - eventuali **adattamenti** alla situazione particolare
  - le nozioni riportate nel **corso frontale**
  - `man scanf` (da terminale)

# La funzione `scanf()`

- `scanf()` **legge** un **input** inserito da un **utente**
- La struttura dell'**input non** è nel pieno **controllo** del programmatore
  - **Esempio:** l'utente usa una **virgola** per **separare due input** destinati alla stessa chiamata di `scanf()`
  - **Esempio:** l'utente inserisce un `float` dove andrebbe un `int`

- In seguito, **consigli generali**. Considerare **sempre**:
  - eventuali **adattamenti** alla situazione particolare
  - le nozioni riportate nel **corso frontale**
  - `man scanf` (da terminale)

## Terminologia

- **Stringa di composizione**: quella **compresa** nella coppia di **doppi apici**
- **Spaziature**: spazi bianchi, i ritorni a capo, le tabulazioni, ecc.
- **Conversioni**: quelle che iniziano con `%`
  - **Esempio:**
    in `scanf("%d %f",&i,&h) ;` la stringa di composizione è `%d %f`,
    nella quale `%d` e `%f` sono due conversioni, separate da una spaziatura (qui lo spazio bianco)

# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )

# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )

In lettura.c

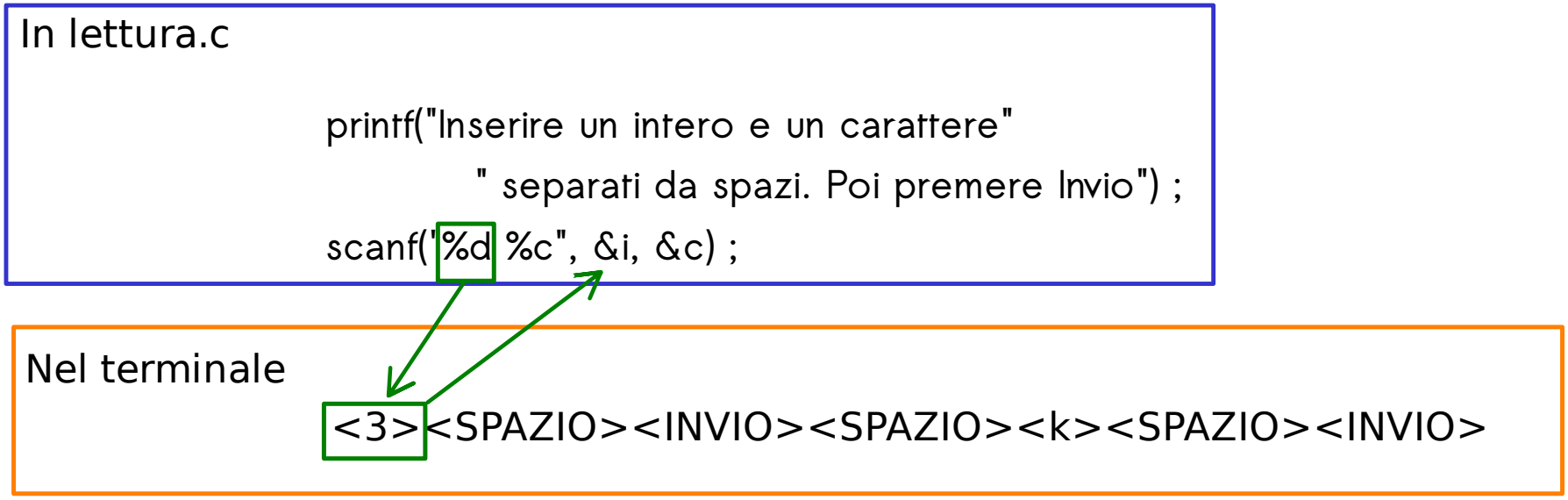
```
printf("Inserire un intero e un carattere"
        " separati da spazi. Poi premere Invio") ;
scanf("%d %c", &i, &c) ;
```

Nel terminale

<3><SPAZIO><INVIO><SPAZIO><k><SPAZIO><INVIO>

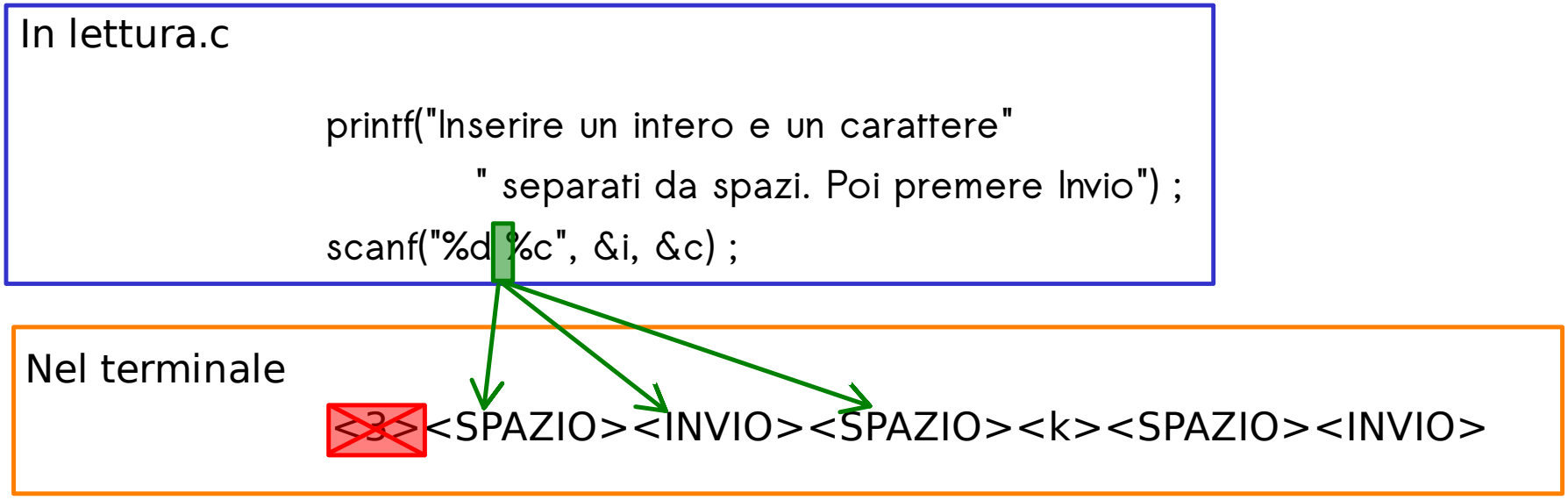
# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )



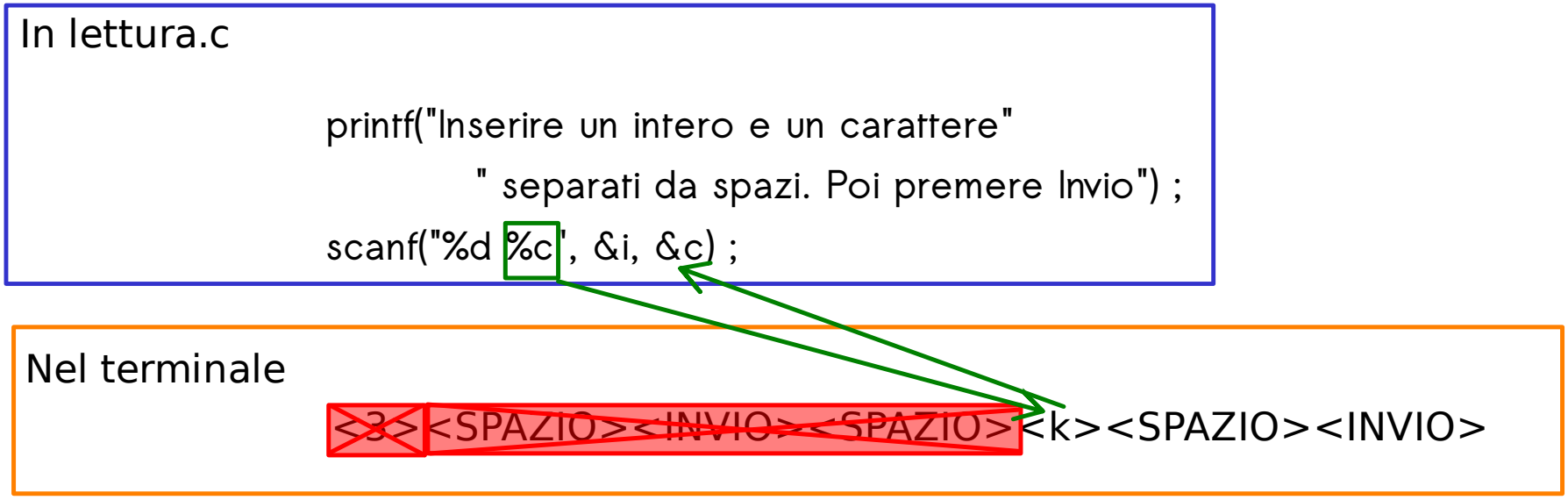
# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )



# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )



# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )

In lettura.c

```
printf("Inserire un intero e un carattere"
        " separati da spazi. Poi premere Invio") ;
scanf("%d %c", &i, &c) ;
```

Nel terminale

<3><SPAZIO><INVIO><SPAZIO><k><SPAZIO><INVIO>

**ANCORA PRESENTI!!**

N.B.
La precedente chiamata a scanf()
ignora gli spazi, quindi NON li salva
nella variabile c.

# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )

In lettura.c

```
printf("Inserire un intero e un carattere"
        " separati da spazi. Poi premere Invio") ;
scanf("%d%c", &i, &c) ;
```

Nota:
Probabile imprecisione del programmatore: dimentica lo spazio tra %d e %c

Nel terminale

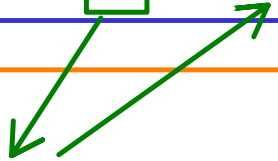
<3><SPAZIO><k><INVIO>

# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )

In lettura.c

```
printf("Inserire un intero e un carattere"
        " separati da spazi. Poi premere Invio") ;
scanf("%d%c", &i, &c) ;
```

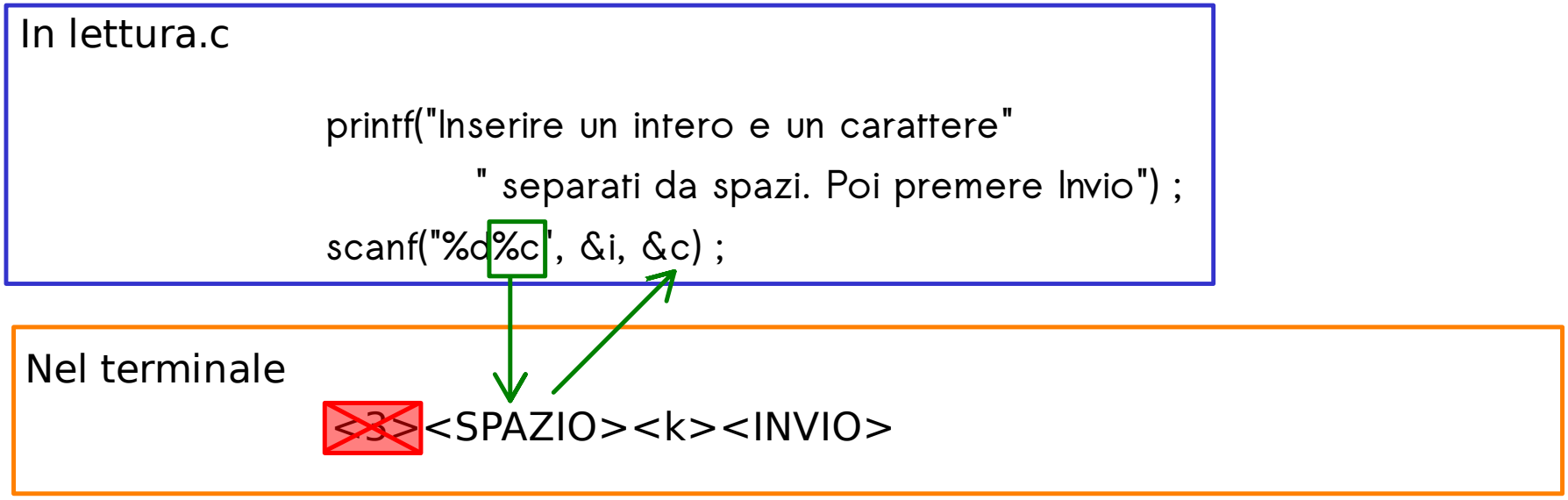


Nel terminale

<3><SPAZIO><k><INVIO>

# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )



# La funzione `scanf()`: dove sorgono i problemi?

- Alla **base** di molti **problemi** c'è la **mancata consapevolezza** di quanto segue:
  - **`Tutto ciò che viene digitato e non letto subito sarà letto in seguito`**
  - **`Per scanf() una spaziatura coincide con un numero arbitrario di spazi bianchi`** (anche nessuno; cfr. **`man scanf`** )

In lettura.c

```
printf("Inserire un intero e un carattere"
        " separati da spazi. Poi premere Invio") ;
scanf("%d%c", &i, &c) ;
```

Nel terminale

<3><SPAZIO><k><INVIO>

**ANCORA PRESENTI!!**

N.B.
La precedente chiamata a scanf() non ignora gli spazi,
quindi obbliga a scrivere "tutto attaccato".
Ad esempio, nel terminale digitare: 3k<INVIO>
La variabile i conterrà il valore 3 e c conterrà il carattere 'k'

# La funzione `scanf()`: linee guida

**Suggerimenti generali (esempi e dettagli nelle slide successive):**

- **Esplicitare** le **modalità di inserimento** dell'input (con `printf()`)
  **Esempio:**

```
printf("Inserire due interi, separati da uno spazio."
"Poi premere Invio\n") ;
```

# La funzione `scanf()`: linee guida

**Suggerimenti generali (esempi e dettagli nelle slide successive):**

- **Esplicitare** le **modalità di inserimento** dell'input (con `printf()`)
- In stringa di composizione usare **solo conversioni** e **spaziature** (non virgole, punti, ecc..)
  **Esempio (EVITARE la virgola tra i %d):**

```
scanf("%d, %d", &i, &h) ;
```

# La funzione `scanf()`: linee guida

**Suggerimenti generali (esempi e dettagli nelle slide successive):**

- **Esplicitare** le **modalità di inserimento** dell'input (con `printf()`)
- In stringa di composizione usare **solo conversioni** e **spaziature** (non virgole, punti, ecc..)
- Usare le **conversioni adeguate** (`%d` per interi, `%f` o `%g` per i float, `%lf` per i double, `%c` per i caratteri)

# La funzione `scanf()`: linee guida

**Suggerimenti generali (esempi e dettagli nelle slide successive):**

- **Esplicitare** le **modalità di inserimento** dell'input (con `printf()`)
- In stringa di composizione usare **solo conversioni** e **spaziature** (non virgole, punti, ecc..)
- Usare le **conversioni adeguate** (`%d` per interi, `%f` o `%g` per i float, `%lf` per i double, `%c` per i caratteri)
- Inserire uno **spazio bianco tra due conversioni** consecutive (**specie se** leggete **caratteri**)

# La funzione `scanf()`: linee guida

**Suggerimenti generali (esempi e dettagli nelle slide successive):**

- **Esplicitare** le **modalità di inserimento** dell'input (con `printf()`)
- In stringa di composizione usare **solo conversioni** e **spaziature** (non virgole, punti, ecc..)
- Usare le **conversioni adeguate** (`%d` per interi, `%f` o `%g` per i float, `%lf` per i double, `%c` per i caratteri)
- Inserire uno **spazio bianco tra due conversioni** consecutive (**specie se** leggete **caratteri**)
- **Non** usare **spaziature alla fine** della stringa di composizione
  **Esempi**
  **EVITARE il `\n`:** `scanf("%d\n", &i) ;`
  **EVITARE lo spazio bianco alla fine della stringa di composizione:**
  `scanf("%d ", &i) ;`

# La funzione `scanf()`: linee guida

**Suggerimenti generali (esempi e dettagli nelle slide successive):**

- **Esplicitare** le **modalità di inserimento** dell'input (con `printf()`)
- In stringa di composizione usare **solo conversioni** e **spaziature** (non virgole, punti, ecc..)
- Usare le **conversioni adeguate** (`%d` per interi, `%f` o `%g` per i float, `%lf` per i double, `%c` per i caratteri)
- Inserire uno **spazio bianco tra due conversioni** consecutive (**specie se** leggete **caratteri**)
- **Non** usare **spaziature alla fine** della stringa di composizione
- **Tra due chiamate** a funzioni di **input** (`scanf()`, `getchar()`, ecc.) "**consumare**" il carattere `\n` prodotto quando si preme `Invio`, **specie se** la **seconda chiamata** legge **caratteri**
  - Va bene anche `%*c` a fine stringa di composizione (vd. slide lezione)

# La funzione `scanf()`: linee guida

**Suggerimenti generali (esempi e dettagli nelle slide successive):**

- **Esplicitare** le **modalità di inserimento** dell'input (con `printf()`)
- In stringa di composizione usare **solo conversioni** e **spaziature** (non virgole, punti, ecc..)
- Usare le **conversioni adeguate** (`%d` per interi, `%f` o `%g` per i float, `%lf` per i double, `%c` per i caratteri)
- Inserire uno **spazio bianco tra due conversioni** consecutive (**specie se** leggete **caratteri**)
- **Non** usare **spaziature alla fine** della stringa di composizione
- **Tra due chiamate** a funzioni di **input** (`scanf()`, `getchar()`, ecc.) "**consumare**" il carattere `\n` prodotto quando si preme `Invio`, **specie se** la **seconda chiamata** legge **caratteri**
  - Va bene anche `%*c` a fine stringa di composizione (vd. slide lezione)

  **Esempio:**

```
scanf("%d", &i) ;
getchar() ;
scanf("%c", &c) ;
```

# La funzione `scanf()`: esempi

**Ipotesi**

- `int i ; char c ; double r ;`

**Lettura separata di un `int` e di un `char`**

```
scanf("%d", &i) ;
getchar() ;
scanf("%c", &c) ;
```

**Lettura simultanea di un `int` e di un `char`**

```
scanf("%d %c", &i, &c) ;
```

**Lettura simultanea di un `int`, di un `double` di un `char`**

```
scanf("%d %lf %c", &i, &r, &c) ;
```

# La funzione `scanf()`: fenomeni paranormali

- Chiede un **input aggiuntivo** dopo `Invio`

```
scanf("%d %lf %c\n", &i, &r, &c) ;
```

  - **Spiegazione:** una **spaziatura nella stringa di composizione** rappresenta a una **sequenza arbitraria di spaziature** consecutive in **quel punto di input**
  - **Soluzione:** togliere `\n` dalla stringa di composizione

# La funzione `scanf()`: fenomeni paranormali

- Chiede un **input aggiuntivo** dopo `Invio`

```
scanf("%d %lf %c\n", &i, &r, &c) ;
```

  - **Spiegazione:** una **spaziatura nella stringa di composizione** rappresenta a una **sequenza arbitraria di spaziature** consecutive in **quel punto di input**
  - **Soluzione:** togliere `\n` dalla stringa di composizione

- **Impossibilità di separare** i caratteri in input

```
scanf("%c%c", &c, &c) ;
```

  - **Spiegazione:** lo **spazio** e il ritorno **a capo** sono **caratteri**, quindi vanno nel secondo argomento
  - **Soluzione:** **spazio** tra conversioni `scanf("%c %c", &c, &c) ;`

# La funzione `scanf()`: fenomeni paranormali

- Chiede un **input aggiuntivo** dopo `Invio`

```
scanf("%d %lf %c\n", &i, &r, &c) ;
```

  - **Spiegazione:** una **spaziatura nella stringa di composizione** rappresenta a una **sequenza arbitraria di spaziature** consecutive in **quel punto di input**
  - **Soluzione:** togliere `\n` dalla stringa di composizione

- **Impossibilità di separare** i caratteri in input

```
scanf("%c%c", &c, &c) ;
```

  - **Spiegazione:** lo **spazio** e il ritorno **a capo** sono **caratteri**, quindi vanno nel secondo argomento
  - **Soluzione:** **spazio** tra conversioni `scanf("%c %c", &c, &c) ;`

- Il **secondo** `scanf()` **non funziona** come atteso

```
scanf("%d %lf", &i, &r) ;
scanf("%c", &c) ;
```

  - **Spiegazione:** il ritorno **a capo** prodotto da `Invio` (primo `scanf()`) viene **letto come carattere** dal secondo `scanf()`
  - **Soluzione:** usare `getchar()` tra i 2 `scanf()`