

# Reti informatiche cod. 545II [9 CFU]

Corso di Laurea in Ingegneria Informatica

---

**Laboratorio e Programmazione di Rete**  
A. A. 2021/2022

Francesco Pistolesi, PhD  
Dipartimento di Ingegneria dell'Informazione  
[francesco.pistolesi@unipi.it](mailto:francesco.pistolesi@unipi.it)

# Lezione 4

Programmazione distribuita in C  
Parte 1

# Introduzione al linguaggio C

# Trova le differenze



C vs. C++

# Chi è?

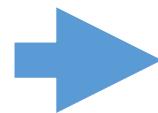


# Definizione di variabili

- Possono essere definite **solo all'inizio di un blocco** (standard C89):

Linguaggio C++

```
int main() {
    int a = 5, i;
    a = func(1000);
    int b = f(a);    ↑
    // ...
    for (i=0; a < 100; i++) {
        b = f(a);
        int c = 0;    ↑
        // ...
    }
}
```



Linguaggio C

```
int main() {
    int a = 5, i;
    int b = f(a);
    a = func(1000);
    // ...
    for (i = 0; a < 100; i++) {
        int c = 0;
        b = f(a);
        // ...
    }
}
```

# Strutture

- Si deve inserire sempre '**struct**' quando si crea una variabile di tipo struttura:

Linguaggio C++

```
struct Complex {  
    double Re;  
    double Im;  
};  
  
int main() {  
    int a = 4;  
    Complex c;  
    // ...  
}
```

Linguaggio C

```
struct Complex {  
    double Re;  
    double Im;  
};  
  
int main() {  
    int a = 4;  
    struct Complex c;  
    // ...  
}
```

# Memoria dinamica (heap)

man malloc  
man 3 free

Si gestisce con le funzioni ***malloc()*** per l'allocazione,  
e ***free()*** per la deallocazione

```
#include <stdlib.h>

int main() {
    int dimensione = 5;
    void* punt;
    punt = malloc(dimensione);
    if (punt == NULL) {
        // Gestione errore
    }
    // ...
    free(punt);
}
```

alloca un'area di memoria di 5 byte che sarà puntata da *punt*

rilascia l'area di memoria di 5 byte

# Allocazione con coercizione

```
#include <stdlib.h>

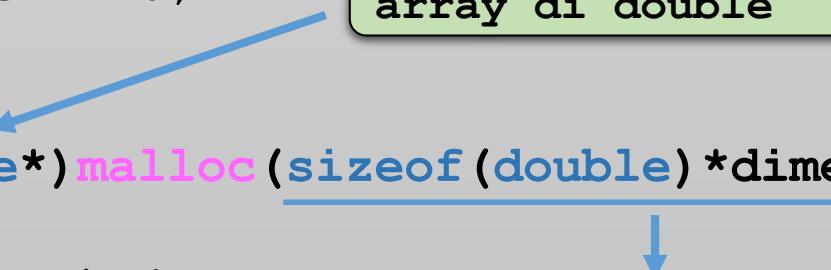
int main() {
    int dimensione = 10;
    double* punt;

    punt = (double*)malloc(sizeof(double)*dimensione)

    if (punt == NULL) {
        // Gestione errore
    }
    // ...
    free(punt);
}
```

l'area di memoria  
è trattata come  
array di double

numero di byte da  
allocare (8\*10=80)



# Ingresso–uscita (libreria stdio.h)

## Gestione dell'uscita

```
int printf(char* formato, lista_argomenti)
```

Stampa **lista\_argomenti** sullo standard-output (video) nel **formato** ricevuto come primo parametro.

Restituisce il numero di caratteri stampati

NOTA: Se si usa, per esempio, "%3.2f" si stampa un valore in virgola mobile con 3 cifre e 2 cifre di precisione (dopo la virgola)

Formato	Cosa viene stampato
%d, %i	Intero decimale
%f	Valore in virgola mobile
%c	Un carattere
%s	Una stringa di caratteri
%o	Numero ottale
%x, %X	Numero esadecimale
%u	Intero senza segno
%f	Numero reale (float o double)
%e, %E	Formato scientifico
%%	Carattere '%'

# Esempi di uscita

```
#include <stdio.h>

char* str = "Com'è bello il C!\n";
// Es. 1
printf(str);
// Es. 2
printf("Messaggio:\n %s", str);
// Es. 3
printf("Da oggi pane e C!");

int i = 5;
// Es. 4
printf("i = %d\n", i);
```

// Es. 1

> Com'è bello il C!

-

// Es. 2

> Messaggio:  
Com'è bello il C!

-

// Es. 3

> Da oggi pane e C!

-

// Es. 4

> i = 5

-

# Ingresso–uscita (libreria stdio.h)

## Gestione dell'ingresso

```
int scanf(char* formato, lista_indirizzi)
```

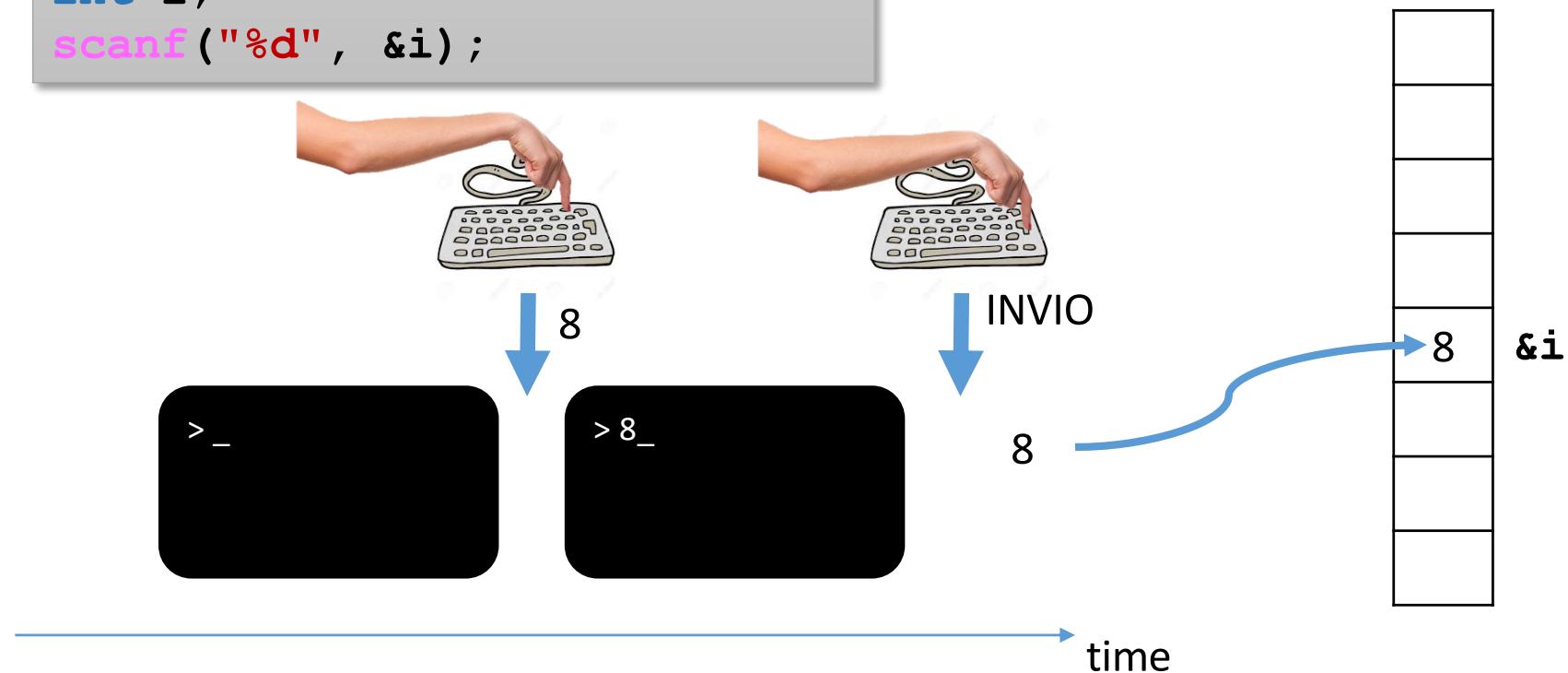
Legge dallo standard-in (tastiera) una sequenza di caratteri (cifre o lettere) e li memorizza agli indirizzi ricevuti in **lista\_indirizzi** nel **formato** ricevuto come primo parametro.

# Esempio di ingresso

man 3 printf  
man scanf  
man stdio

```
#include <stdio.h>

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



# Stringhe: dimensione e confronto

man **strlen**  
man **strcmp**

```
#include <string.h>

// Lunghezza
char* str1 = "Gatto \n";
int len;
len = strlen(str1);

// Confronto
char* str2 = "Gattone \n";
int ret;
ret = strcmp(str1, str2);
```



8 byte allocati, ma **strlen()** restituisce 7!

**strcmp()** confronta le due stringhe un carattere alla volta



**ret = 0** se le stringhe sono identiche  
**ret < 0** se il primo carattere di **str1** diverso dal corrispondente di **str2** ha codifica ASCII minore  
**ret > 0** viceversa

# Stringhe: copia e concatenazione

man strncpy  
man strncat

```
#include <string.h>

// Copia
char str1[20];
n = sizeof(str1);
strncpy(str1, "Gatto \n", n);

// Concatenazione
char* str2 = "nero\n";
strncat(str1, str2, 6);
```



- Fare attenzione a **copiare anche il terminatore ('0')**
- Le funzioni **non controllano che ci sia spazio** nella stringa di destinazione



# Gestione dei file

man fopen

```
#include <stdio.h>

// Apertura file
FILE* fd;
fd = fopen("/tmp/gatto.txt", "r" );
if (fd == NULL) {
    // Gestione errore
}
```

Si specifica un **percorso** e una **modalità** di apertura:

- **r** = sola lettura
- **w** = sola scrittura
- **r+** = lettura e scrittura
- **a** = *append*
- **a+** = lettura e *append*



Se si specifica scrittura o append e il file non esiste,  
il file viene creato

# Gestione dei file: *fscanf()* ed *fprintf()*

```
#include <stdio.h>

// Lettura da file
int ret, n;
FILE* fd1;
fd1 = fopen("/tmp/gatti.txt", "r");
ret = fscanf(fd1, "%d", &n);

// Scrittura su file
char* str = "Gatto!\n";
FILE* fd2;
fd2 = fopen("/tmp/gatti2.txt", "w");
ret = fprintf(fd2, "%s", str);
```

**man fscanf**  
**man fprintf**

Funzionano come **scanf()** e  
**printf()**, ma usano  
il file specificato anziché  
**stdin** e **stdout**

# Compilazione

man gcc

Useremo la *GNU Compiler Collection (GCC)*

Con la **compilazione**, si creano i *file oggetto* a partire dai *file sorgente*

Con il **linking**, si crea un *file eseguibile* a partire dai *file oggetto*

Esecuzione

