

# Programmazione - lezione 5

Appunti di Davide Vella 2024/2025

Alessandro Mazzei

[alessandro.mazzei@unito.it](mailto:alessandro.mazzei@unito.it)

Link al moodle :

[informatica.i-learn.unito.it/course/view.php?id=2982](http://informatica.i-learn.unito.it/course/view.php?id=2982)

**03/10/2024**

## Contenuti

1. [Scambio di due variabili](#)
2. [Float e Double](#)
3. [Math](#)
4. [Conversioni di tipo](#)
  - 4.1 [Conversioni implicite](#)
  - 4.2 [Conversione esplicita](#)
5. [Indirizzi di memoria](#)
6. [Funzioni](#)
  - 5.1 [Parametri](#)
  - 5.2 [Restituire](#)
    - 5.2.1 [Void](#)
      - 5.2.1.1 [Printf\(\) e scanf\(\)](#)
7. [Scanf\(\), input da tastiera](#)
8. [Puntatori](#)
9. [Booleani](#)

## Scambio di due variabili

Si usa una variabile temporanea, il perché è il seguente :

nome variabile	valore
x	5
y	2

se facciamo  $x = y$  o  $y = x$  incontreremmo un problema, ovvero uno dei due valori scomparirebbe, prendiamo in caso  $x = y$  :

x	2
y	2

Vediamo che il 5 non esiste più. Allora usiamo una terza variabile e facciamo  $tmp = x$ ,  $x = y$ ,  $y = tmp$  e avremo :

x	2
y	5
tmp	2

Abbiamo quindi scambiato il valore di due variabili tra di loro.

```
int x = 5;  
int y = 2;  
int tmp;  
tmp = x;  
x = y;  
y = tmp;
```

## Float e Double

Float e Double sono due tipi di variabili basiche di c. Questi due tipi possono tenere entrambi dei numeri con la virgola, con la differenza che float occupa 4 byte (quindi  $3.4^{38}$  numeri positivi e altri  $3.4^{38}$  numeri negativi), mentre double occupa 8 byte (quindi  $1.7 \times 10^{308}$  numeri positivi e altri  $1.7 \times 10^{308}$  numeri negativi). Per prendere in input o stampare in output dei float si usa il

`%f`

Nel caso della stampa di un float con quanto scritto sopra otteniamo come risultato qualcosa del genere :

 **Exemple**

2.500000

Se invece volessimo approssimare le il numero (e rimuovere tutti gli zero che non vogliamo) possiamo fare

```
%.2f
```

Questo indica che dopo il punto ci saranno  $n$  numeri (in questo caso 2 numeri dopo il punto).

 **Exemple**

2.50

Mentre per prendere in input o stampare in output dei double si usa il

```
%lf
```

## Math

La libreria `<math.h>` è una libreria di c che permette di aggiungere funzioni matematiche extra rispetto alle 5 operazioni algebriche base (+, -, ×, /, %). Alcuni esempi di funzioni aggiuntive sono :

- cos
- sin
- sqrt
- pow (potenze).

## Conversioni di tipo

### Conversioni implicite

Le conversioni implicite sono tutte quelle conversioni che vengono effettuate automaticamente quando avviene un'assegnazione tramite operazione da un tipo ad un altro.

```
int x = 5;  
int y = 2;  
float z;  
z = y/x;
```

Il risultato di questa operazione potrebbe risultare scontato (2.5), ma in realtà stiamo dividendo 2 interi tra di loro, quindi il risultato viene 2.000000. Questo avviene perché il programma non tiene conto della variabile in cui stiamo assegnando il risultato dell'operazione (un float in

questo caso, ovvero una variabile che può avere numeri dopo la virgola e che quindi potrebbe tenere 2.5), ma guarda unicamente il tipo delle due variabili con cui stiamo facendo l'operazione. Detto questo, basta che una delle due variabili sia un float o un double per avere come risultato 2.5 :

```
int x = 5;
float y = 2;
float z;
z = y/x;
```

## Conversione esplicita

La conversione esplicita (detta comunemente "cast") è quella che permette di "forzare" una rappresentazione di formato anche se ne stiamo usando due diversi. Il cast si fa inserendo le parentesi tonde e dentro ad esse il tipo in cui vogliamo castare. Se guardiamo il caso di sopra possiamo quindi fare :

```
int x = 5;
int y = 2;
float z;
z = (float) y/x;
```

Per ottenere come risultato 2.5 senza aver bisogno che una delle due variabili sia un float/double.

## Indirizzi di memoria

Gli indirizzi sono rappresentati sotto forma di numeri esadecimali.

Una variabile di per se (si può vedere 'x' sopra per esempio) è una specie di contenitore che tiene dentro di se un dato valore, ma questa variabile si trova in un luogo preciso della memoria, un indirizzo. Per ottenere l'indirizzo di quella memoria utilizziamo quindi il simbolo '&' davanti al nome della variabile di cui vogliamo l'indirizzo. Vediamo subito un esempio :

```
int x = 5;
printf("x vale %d ed e' salvata nell'indirizzo %p", x, &x);
```

L'output che otteniamo è questo : **x vale 5 ed e' salvata nell'indirizzo 0061FF1C**.

Come si può vedere otteniamo prima il valore di x e poi la prima cella di memoria dove è salvata la variabile x. X è un int, non occupa una singola cella di memoria, ma ne occupa più di

una (4 byte), quindi, se pensiamo che la maggior parte della macchine ragionano con i byte (definizione informale che serve a far capire il ragionamento), noi con '&x' otteniamo il primo byte in cui è salvato 'x' e possiamo leggere i 3 byte successivi per avere di nuovo il valore di 'x'. Vediamo un esempio di ciò con il seguente codice :

```
int x = 5;

int y = 2;

int z = 3;

printf("x vale %d ed e' salvata nell'indirizzo %p\n", x, &x);

printf("y vale %d ed e' salvata nell'indirizzo %p\n", y, &y);

printf("z vale %d ed e' salvata nell'indirizzo %p\n", z, &z);
```

L'output del codice è il seguente :

```
x vale 5 ed e' salvata nell'indirizzo 0061FF1C
y vale 2 ed e' salvata nell'indirizzo 0061FF18
z vale 3 ed e' salvata nell'indirizzo 0061FF14
```

Possiamo chiaramente vedere che gli indirizzi di memoria sono contigui e si differenziano di 4 tra di loro.

## Funzioni

### Warning

Attenzione, questa parte non è stata ancora spiegata dal professore, ma ritengo che possa, spero, facilitare la comprensione di argomenti futuri come printf() e scanf(). Leggere quindi questo paragrafo è totalmente opzionale e non obbligatorio per capire gli argomenti successivi (scanf()) e input da tastiera.

Tutti i programmi in C sono formati da funzioni. Il main stesso è una funzione (particolare visto che è l'unica funzione che dev'essere presente per forza per far sì che un programma sia eseguibile<sup>1</sup>). Una funzione è quindi una parte di codice separata dal main che ha una precisa funzione e che può restituire un risultato (non per forza). Vediamo un esempio :

```
#include <stdio.h>

int somma(int n1, int n2) {
    int somma = n1 + n2;
```

```

        return somma;
    }

    int main() {
        int n1 = 5;
        int n2 = 2;
        int somma = somma(n1, n2);
        printf("La somma di n1 e n2 equivale a %d", somma);
        return 0;
    }

```

Come possiamo vedere, nel codice sopra c'è il solito main che usiamo in tutti i nostri programmi e poi c'è la funzione somma, una funzione che prende due numeri come *parametro*, ne fa la somma e *restituisce* il risultato.

## Parametri

I parametri sono semplicemente delle variabili o delle costanti (tipo 1, 2, 5, "ciao"...) che vengono date alla funzione quando vengono chiamate da altre funzioni (come il main stesso). Questi parametri possono essere poi usati nella funzione per fare qualcosa (come la somma ad esempio). Tutti i parametri sono divisi da una virgola, come vediamo sopra, dobbiamo passare due parametri (n1 e n2) e quindi mettiamo una virgola tra questi due.

## Restituire

Le funzioni possono restituire un valore. Il tipo del valore che viene restituito dalla funzione lo si può capire dalla parola che precede il nome della funzione. Sia nella funzione di esempio che nel main entrambe le funzioni restituiscono un int. Al posto di int ci potrebbe essere un qualsiasi altro tipo, come "long, double, float, long long, char...".

## Void

Una funzione che non restituisce nulla si indica con void (prima del nome della funzione). Queste funzioni sono usate in casi e in modo attualmente complicati da spiegare, ma possiamo vedere due esempi chiari che conosciamo già : printf() e scanf().

## Printf() e scanf()

Printf() e scanf() sono due funzioni che non restituiscono nulla, infatti quando le usiamo non assegniamo mica il risultato ad una variabile (anche perché il risultato è inesistente). Per chiarire (spero) ancora meglio farò un esempio.

```
#include <stdio.h>
int somma(int n1, int n2) {
    int somma = n1 + n2;
    return somma;
}
void moltiplicazione(int n1, int n2){
    int moltiplicazione = n1 * n2;
    return moltiplicazione;
}

int main() {
    int n1 = 5;
    int n2 = 2;
    int somma = somma(n1, n2);
    moltiplicazione(n1, n2);
    printf("La somma di n1 e n2 equivale a %d", somma);
    return 0;
}
```

Come possiamo vedere somma restituisce un intero (la somma di n1 e n2) e moltiplicazione non restituisce nulla (in questo caso risulta anche inutile, ma non ce ne preoccupiamo ora).

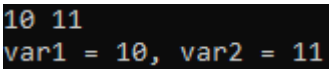
## Scanf(), input da tastiera

Scanf() è la funzione che useremo per prendere in input da tastiera una qualsiasi cosa che vogliamo usare nel programma. La sintassi di scanf è la seguente :

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

Quello che stiamo facendo nel pezzo di codice qua sopra, per quanto sembra difficile, è in realtà molto facile. Chiamiamo la funzione scanf() e mettiamo **sempre** una stringa (ovvero una scritta chiusa tra due doppi apici "abc"). In questa stringa dobbiamo mettere il % e poi mettiamo la lettera adeguata alla variabile in cui vogliamo salvare l'input dell'utente (negli appunti della lezione 3 c'è una tabella che può aiutare per sapere cosa mettere in caso di molte variabili). Dopo questo dobbiamo passare l'indirizzo di memoria della variabile che vogliamo sovrascrivere (più semplicemente, come abbiamo visto prima, mettiamo la '&' davanti al nome della variabile in cui dobbiamo scrivere il valore). Possiamo anche fare :

```
int var1;
int var2;
scanf("%d %d", var1, var2);
printf("var1 = %d, var2 = %d", var1, var2);
```

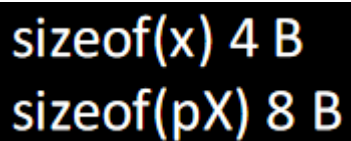
Questo ne sarebbe il risultato :  in questo caso inseriamo un numero, lasciamo un spazio, inseriamo un altro numero e poi premiamo invio.

## Puntatori

I puntatori sono variabili i cui valori sono indirizzi di memoria. Un puntatore (pointer), quindi, è una variabile che non contiene dati di tipo primitivo, bensì un indirizzo di un'altra variabile che contiene un valore specifico. Il puntatore punta a quella variabile. Spesso è necessario memorizzare in un puntatore l'indirizzo ritornato dall'operatore di indirizzo '&'. Ogni tipo primitivo ha un tipo puntatore corrispondente indicato come il tipo seguito dall'asterisco (*star*). *Un 'puntatore a variabile di tipo float' sarà codificato come float.*

Ecco un piccolo esempio di utilizzo dei puntatori :

```
int main() {
    int x;
    int* pX;
    printf("sizeof(x) %d B\n",
        sizeof(x));
    printf("sizeof(pX) %d B\n",
        sizeof(pX));
}
```



```
sizeof(x) 4 B
sizeof(pX) 8 B
```

## Booleani

I booleani sono dei tipi di dati i cui unici due possibili sono indicati con "true" e "false" oppure come 1 e 0. Data la mancanza del tipo "bool" in C, useremo queste cose solo quando vedremo gli if, per cui, si potrebbe rimandare questa spiegazione sino a quel punto, in caso contrario : i booleani sono "generati" a partire da un'espressione booleana (un'espressione binaria, un'espressione che si esegue solamente tra 2 elementi), per rendere più chiaro il ciò, vediamo un



esempio che rende tutto più semplice :

$$a < b$$

Il risultato di questa espressione restituisce (come detto prima) :

- 1 se è vero che  $a$  è minore di  $b$
- 0 se è falso che  $a$  è minore di  $b$

Gli operatori che restituiscono un booleano sono questi :

- `|`  $(a > b)$
- `<`  $(a < b)$
- `|`  $= (a >= b)$
- `<=`  $(a <= b)$
- `==`  $(a == b, \text{ovvero chiediamo se } a \text{ ha lo stesso valore di } b)$
- `!=`  $(a != b, \text{ovvero chiediamo se } a \text{ è diverso da } b)$

Se prendiamo come esempio il seguente codice :

```
#include <stdio.h>

int main() {

    printf("%d\n", 3 > 4);

    printf("%d\n", 3 >= 4);

    printf("%d\n", 3 == 4);

    printf("%d\n", 3 != 4);

    printf("%d\n", 3 <= 4);

    printf("%d\n", 3 < 4);

}
```

Il risultato è :

```
0
0
0
1
1
1
```

---

<sup>1</sup> : Esistono anche file .c che non servono per essere eseguiti come li stiamo affrontando adesso.