

Linguaggio C:

struttura di un programma
variabili e tipi di dati semplici
operatori
funzioni di input-output

Università degli Studi di Brescia

Docente: Massimiliano Giacomini

Struttura di un programma C

- Un semplice esempio: programma che produce a video la scritta “Ciao mondo”

```
#include<stdio.h>  
#include <stdlib.h>
```

} *Parte dichiarativa globale*

```
main(){  
    printf("Ciao mondo\n");  
    system("pause");  
}
```

} *Programma principale*

Altre funzioni

Parte dichiarativa globale

- Nel nostro caso, conterrà sempre direttive al compilatore e in particolare la direttiva *#include*:

`#include<stdio.h>`

- Prima che il file sia compilato, viene sostituita dal file *stdio.h*, che include le dichiarazioni delle funzioni della libreria standard di I/O
- In pratica, la direttiva è necessaria per poter usare le “istruzioni” (che in realtà sono funzioni) di ingresso e uscita
- Notare che le direttive non sono seguite dal punto e virgola

Programma principale

- Parte dichiarativa: vengono dichiarate le *variabili* (e le costanti) usate dal programma
- Parte esecutiva: la sequenza delle *istruzioni* del programma
 - le istruzioni sono normalmente eseguite in sequenza
 - ciascuna istruzione termina con un punto e virgola
- Tre tipologie di istruzioni principali:
 - istruzioni ingresso/uscita:
permettono di acquisire dati e produrre risultati
 - istruzioni aritmetico/logiche
permettono di svolgere calcoli
 - istruzioni di controllo
permettono di alterare il flusso sequenziale di esecuzione delle istruzioni

Rivediamo il programma precedente

```
#include<stdio.h>
```

```
#include <stdlib.h>
```

```
main(){
```

```
    printf("Ciao mondo\n");
```

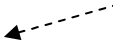
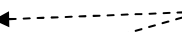
```
    system("pause");
```

```
}
```

La parte dichiarativa è nulla



Istruzioni di ingresso/uscita



- Un altro esempio con parte dichiarativa non nulla

```
#include<stdio.h>
```

```
main(){
```

```
    int x;
```

```
    int y, z;
```

```
    x=4;
```

```
    y=5;
```

```
    z=x+y;
```

```
    x=x+1;
```

```
    x=x+y+z;
```

```
}
```

} Parte dichiarativa

} Parte esecutiva

Parte dichiarativa di un programma

- In questa parte vengono “dichiarate” le *variabili* e le *costanti* usate nel programma
- Notare che le dichiarazioni precedono le istruzioni eseguibili, quindi tutte le variabili e le costanti sono dichiarate prima del loro uso

Variabili

- Una variabile rappresenta una “porzione di memoria” per contenere un valore
- Il valore di una variabile può essere cambiato durante l’esecuzione del programma, mediante un’operazione detta *assegnamento*

Rivediamo il programma precedente

```
#include<stdio.h>
```

```
main(){
```

```
    int x;
```

```
    int y, z;
```

```
    x=4;
```

```
    y=5;
```

```
    z=x+y;
```

```
    x=x+1;
```

```
    x=x+y+z;
```

```
}
```

x

y

z

Rivediamo il programma precedente

```
#include<stdio.h>
```

```
main(){  
    int x;  
    int y, z;
```

```
    x=4;
```

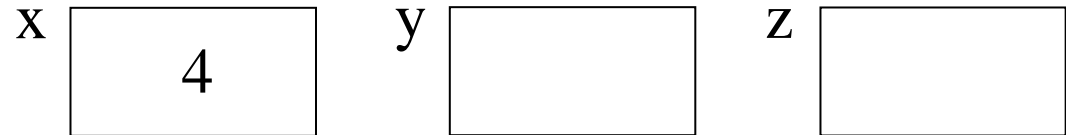
```
    y=5;
```

```
    z=x+y;
```

```
    x=x+1;
```

```
    x=x+y+z;
```

```
}
```



Rivediamo il programma precedente

```
#include<stdio.h>
```

```
main(){  
    int x;  
    int y, z;
```

```
    x=4;
```

```
    y=5;
```

```
    z=x+y;
```

```
    x=x+1;
```

```
    x=x+y+z;
```

```
}
```



Rivediamo il programma precedente

```
#include<stdio.h>
```

```
main(){  
    int x;  
    int y, z;
```

```
    x=4;
```

```
    y=5;
```

```
    z=x+y;
```

```
    x=x+1;
```

```
    x=x+y+z;
```

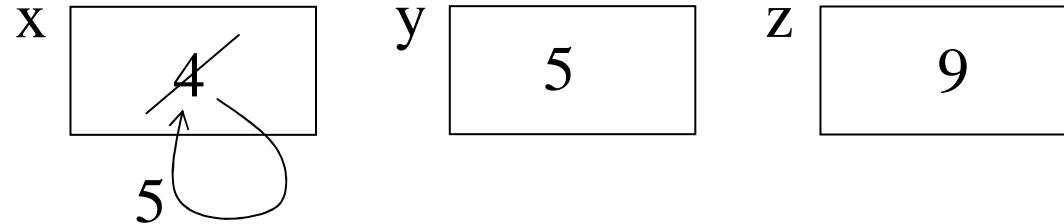
```
}
```



Rivediamo il programma precedente

```
#include<stdio.h>
```

```
main(){  
    int x;  
    int y, z;  
  
    x=4;  
    y=5;  
    z=x+y;  
    x=x+1;  
    x=x+y+z;  
}
```



Rivediamo il programma precedente

```
#include<stdio.h>
```

```
main(){  
    int x;  
    int y, z;
```

```
    x=4;
```

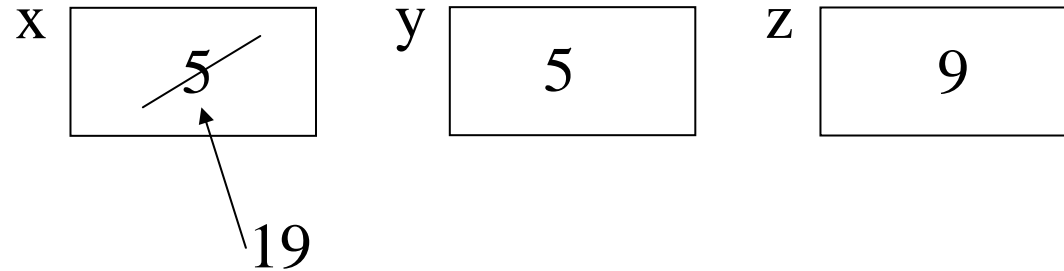
```
    y=5;
```

```
    z=x+y;
```

```
    x=x+1;
```

```
    x=x+y+z;
```

```
}
```

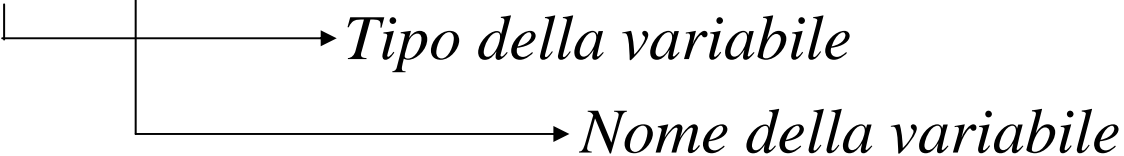


Le caratteristiche di una variabile



- *Nome* e *tipo* devono essere dichiarati
 - > nome: sequenza di caratteri che comincia con un carattere alfabetico, univoco, case sensitive, non può essere una “keyword” (parola riservata del linguaggio C)
 - > tipo: identifica i **valori** che la variabile può assumere e le **operazioni** che su di essa possono essere compiute
- P.es. esistono variabili per memorizzare valori numerici interi, valori numerici reali, dati di una persona, ...

Dichiarazione di variabili: sintassi

`int somma;`

→ *Tipo della variabile*
→ *Nome della variabile*

- E' anche possibile dichiarare più variabili dello stesso tipo:

`int primo, secondo;`

- E' anche possibile dichiarare variabili e assegnarvi dei valori:

`int altezza=3;`

- Negli esempi che faremo la dichiarazione di una variabile coincide con la creazione della variabile stessa

Perché si devono dichiarare tutte le variabili prima di usarle?

```
#include<stdio.h>
```

```
main(){  
    int somma;  
    int x, y;
```

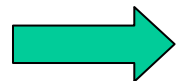
```
    ...
```

```
    somme=x+y;
```

```
    ...
```

```
}
```

Supponiamo che scriva
somme al posto di *somma*



In questo caso il compilatore “se ne accorge”,
perché *somme* non è stata dichiarata!

Tipi di dati in C

	Semplici	Strutturati
Predefiniti	<i>char, int, float, double</i>	-
Definiti dall'utente	<i>Ridefinizione, enum</i>	<i>definiti con: array, struct, pointer</i>

Tipi di dati semplici: le variabili contengono informazioni
“logicamente indivisibili”

Tipi di dati strutturati: informazioni scomponibili in più componenti
(es: i giocatori di una squadra di calcio)

Tipi di dati in C

	Semplici	Strutturati
Predefiniti	<i>char, int, float, double</i>	-
Definiti dall'utente	<i>Ridefinizione, enum</i>	<i>definiti con: array, struct, pointer</i>

In C i tipi di dati semplici predefiniti (disponibili direttamente) sono tutti numerici

Tipi di dati semplici predefiniti

1) Tipi che definiscono “variabili intere”

```

graph LR
    subgraph CHAR
        C1["[signed] char"]
        C2["unsigned char"]
        C1 --- C2
        C1 --- C3["Spazio allocato: 1 byte"]
        C1 --- C4["Signed: -128 < v < 127"]
        C1 --- C5["Unsigned: 0 < v < 255"]
        C1 --- C6["Tipicamente rappresentano caratteri"]
    end
    subgraph INT
        I1["[signed] short int"]
        I2["[signed] int"]
        I3["[signed] long int"]
        I4["unsigned short int"]
        I5["unsigned int"]
        I6["unsigned long int"]
        I1 --- I2 --- I3
        I4 --- I5 --- I6
        I1 --- I7["Spazio allocato (lo stesso per signed e unsigned):"]
        I1 --- I8["spazio(short int) <= spazio(int) <= spazio(long int)"]
        I1 --- I9["Tipicamente:"]
        I1 --- I10["short int: 2 byte (-32768 < v < 32767 o 0 < v < 65535)"]
        I1 --- I11["int e long int: 4 byte (-2,147,483,648 < v < 2,147,483,647"]
        I1 --- I12["o 0 < v < 4,294,967,295)"]
    end
  
```

CHAR

- [signed] char unsigned char
- Spazio allocato: 1 byte
- Signed: $-128 < v < 127$
- Unsigned: $0 < v < 255$
- Tipicamente rappresentano caratteri

INT

- [signed] short int [signed] int [signed] long int
- unsigned short int unsigned int unsigned long int
- Spazio allocato (lo stesso per signed e unsigned):
spazio(short int) \leq spazio(int) \leq spazio(long int)
- Tipicamente:
- short int: 2 byte $(-32768 < v < 32767 \text{ o } 0 < v < 65535)$
- int e long int: 4 byte $(-2,147,483,648 < v < 2,147,483,647$
o $0 < v < 4,294,967,295)$

Operazioni su variabili int e char

=	Assegnamento (int-int o char-char)	}	Risultato: il valore assegnato (int o char)
+	Somma (tra int o char)		
-	Sottrazione (tra int o char)	}	Producono risultato int o char
*	Moltiplicazione (tra int o char)		
/	Divisione (tra int o char) con troncamento della parte frazionaria		
%	Resto (tra int o char) della divisione intera		
==	Relazione di uguaglianza (tra int o char)	}	Producono risultato (int o char): - 0 per “falso” - >0 per “vero”
!=	Relazione di disuguaglianza (tra int o char)		
<	Relazione “minore di” (tra int o char)		
>	Relazione “maggiore di” (tra int o char)		
<=	Relazione “minore o uguale” (tra int o char)		
>=	Relazione “maggiore o uguale” (tra int o char)		

2) Tipi che definiscono “variabili reali”

- I numeri sono rappresentati nel calcolatore in virgola mobile
- In questo caso non abbiamo le varianti signed vs. unsigned:
i numeri sono tutti con segno (cfr. rappresentaz. in virgola mobile)
- I tipi sono due: FLOAT e DOUBLE, quest'ultimo ha anche la variante LONG



float

double

long double

Spazio allocato:

$\text{spazio(float)} \leq \text{spazio(double)} \leq \text{spazio(long double)}$

Tipicamente:

float: 4 byte

double: 8 byte

long double: spesso anche per essi 8 byte!

Operazioni su variabili float e double

=	Assegnamento	}	Risultato: il valore assegnato (float o double)
+	Somma		
-	Sottrazione	}	Producono risultato float o double
*	Moltiplicazione		
/	Divisione a risultato reale		
==	Relazione di uguaglianza	}	Producono risultato intero: - 0 per “falso” - >0 per “vero”
!=	Relazione di disuguaglianza (tra int o char)		
<	Relazione “minore di” (tra int o char)		
>	Relazione “maggiore di” (tra int o char)		
<=	Relazione “minore o uguale” (tra int o char)		
>=	Relazione “maggiore o uguale” (tra int o char)		

COSA DOVETE SAPERE?

- Di fatto, userete i tipi

char, int, float

Questi occorre ricordarseli!

- Facendo un po' di pratica, si imparano gli operatori più comuni
- Per il resto, non occorre ricordare molto, ma capirlo sì!

 Approfondiamo gli operatori, facendo alcuni esempi
“illuminanti” (forse)

Cosa sono gli operatori?

- Ricevono in ingresso uno o più valori di un certo tipo (operandi)
- Restituiscono un valore di un certo tipo (eventualmente diverso)

Esempio

L'operatore aritmetico +

$5 + 2 \longleftarrow$ Restituisce 7

$x + y \longleftarrow$ Restituisce il valore pari a $x+y$

- In C esistono operatori binari (due valori) ed operatori unari
- Per ogni tipo esistono operatori specifici; p.es. il resto % esiste per il tipo *int* ma non per il *float* – diversi operatori possono avere nome uguale (es. la divisione / diversa per *int* e *float*)

Operatore di assegnamento

- Sintassi:

nomevariabile = espressione

- Espressione:

costruita a partire da variabili e/o costanti eventualmente mediante operatori. Più precisamente:

- una variabile o una costante è un'espressione
- date due espressioni E_1 ed E_2 ed un operatore binario $*$,
 $(E_1 * E_2)$ è un'espressione
- data un'espressione E ed un operatore unario $*$,
 $*(E)$ è un'espressione

- Significato dell'operatore di assegnamento:

- > prima l'espressione viene valutata (viene calcolato un valore)
- > poi il valore viene assegnato alla variabile a sinistra

- Esempio

```
int x = 3;  
int y;
```

```
y = (x+2)*3    // y=15  
x = x+1;       // x=4
```

- Come tutti gli operatori, anche l'assegnamento restituisce un valore, ovvero il valore assegnato:

```
int x, y;  
y=(x=3);       //x=3 assegna 3 a x e restituisce 3, assegnato a y:  
               //x=3, y=3
```

Operatori aritmetici

- Operatori binari (già visti in precedenza):
 - + addizione
 - * moltiplicazione
 - sottrazione
 - / divisione (diversa a seconda sia applicata a int o float)
 - % resto (tra interi)
- Operatore unario: segno negativo -
- Esempio:

```
int dividendo=7, divisore=3,quoziente, resto;  
quoziente=dividendo/divisore;           //2  
resto=dividendo%divisore;               //1
```
- Notare che la divisione ha un comportamento diverso a seconda del tipo... vediamo alcuni esempi

- Un esempio: media tra tre numeri interi

```
#include<stdio.h>
#include <stdlib.h>
main(){
    int a, b, c, media;
    a=5;
    b=7;
    c=11;
    media=(a+b+c)/3;
    printf("La media tra %d e %d e %d fa %d\n", a,b,c,media);
    system("pause");
}
```

Quale numero viene stampato?

- Un esempio: media tra tre numeri interi

```
#include<stdio.h>
#include <stdlib.h>
main(){
    int a, b, c, media;
    a=5;
    b=7;
    c=11;
    media=(a+b+c)/3;
    printf("La media tra %d e %d e %d fa %d\n", a,b,c,media);
    system("pause");
}
```

7

- Un esempio: media tra tre numeri interi

```
#include<stdio.h>
#include <stdlib.h>
main(){
    float a, b, c, media;
    a=5;
    b=7;
    c=11;
    media=(a+b+c)/3;
    printf("La media tra %f e %f e %f fa %f\n", a,b,c,media);
    system("pause");
}
```

Quale numero viene stampato?

- Un esempio: media tra tre numeri interi

```
#include<stdio.h>
#include <stdlib.h>

main(){
    float a, b, c, media;
    a=5;
    b=7;
    c=11;
    media=(a+b+c)/3;
    printf("La media tra %f e %f e %f fa %f\n", a,b,c,media);
    system("pause");
}
```

7.666667

Espressioni con operandi di tipo diverso e conversioni implicite

- Se gli operatori sono applicati ad operandi di tipo diverso, il C applica *regole di conversione implicita*: l'espressione è valutata se i tipi sono “compatibili”, ovvero resi uguali da tali regole.
 - ad esempio, ove necessario un valore *int* sarà convertito in *float*, ma un valore di tipo *struct* (es. elemento di una rubrica telefonica) non può essere convertito in un valore di tipo numerico!
- Espressioni del tipo $x \text{ op } y$ (op: operatore aritmetico)
 - le regole generali sono piuttosto articolate
 - dato che voi usate *char*, *int* e *float*, vi basta considerare la conversione $\text{char} \rightarrow \text{int} \rightarrow \text{float}$ (vedi esempio successivo)
- Espressioni di assegnamento: se f è *float* e i è *int*
 - $f = i;$ //valore di i convertito in un *float* e assegnato a f
(non c'è perdita di informazione)
 - $i = f;$ // valore di f convertito in *int* (perdita di informazione se ad esempio f non è un intero)

- Un esempio (poco ragionevole): media tra tre numeri interi

```
#include<stdio.h>
#include <stdlib.h>

main(){
    float a, b, c;
    int media;
    a=5;
    b=7;
    c=11;
    media=(a+b+c)/3;    //valore 7.666667 troncato e assegnato
                        //a media!
    printf("La media tra %f e %f e %f fa %d\n", a,b,c,media);
    system("pause");
}
```

7

- Un esempio: media tra tre numeri interi

```
#include<stdio.h>
#include <stdlib.h>

main(){
    int a, b, c;
    float media;
    a=5;
    b=7;
    c=11;
    media=(a+b+c)/3;    //valore intero 7 assegnato a media!
    printf("La media tra %d e %d e %d fa %f\n", a,b,c,media);
    system("pause");
}
```

7.000000

- Un esempio: media tra tre numeri interi

```
#include<stdio.h>
#include <stdlib.h>

main(){
    int a, b;
    float c;
    float media;
    a=5;
    b=7;
    c=11;
    media=(a+b+c)/3;      //a+b+c: conversione a float e quindi
                          //la divisione è tra float: 7.666667
                          //valore float assegnato a media!
    printf("La media tra %d e %d e %f fa %f\n", a,b,c,media);
    system("pause");
}
```

7.666667

Forme abbreviate (assegnamento e incremento)

- $e1 \text{ op} = e2$ equivale a $e1 = e1 \text{ op } e2$, per esempio:
 $a += b; \quad // a = a + b$
 $a *= b; \quad // a = a * b$
- $++a; \quad // a = a + 1$ (come tale la valutazione dell'espressione $++a$ è $a + 1$, ovvero prima viene incrementata la variabile a poi viene valutata l'espressione)
- $a++; \quad // a = a + 1$, in cui però la valutazione dell'espressione $++a$ è a (prima si valuta l'espressione poi si incrementa a)
- $--a; \quad // a = a - 1$, in cui la valutazione dell'espressione è $a - 1$
- $a--; \quad // a = a - 1$, in cui la valutazione dell'espressione è a

- Esempi su forma prefissa e postfissa

```
#include<stdio.h>
```

```
main(){
```

```
    int x,y;
```

```
    x=5;
```

```
    y=x++;
```

```
    // y=5, x=6
```

```
    // equivale a y=x; x=x+1
```

```
    x=5;
```

```
    y=(x=x+1);
```

```
    // x=6, y=6
```

```
    x=5;
```

```
    y=++x;
```

```
    //x=6, y=6
```

```
    //equivale a x=x+1;y=x
```

```
    ...
```

```
}
```

Operatori relazionali

- Operatori su tipi numerici (già visti in precedenza):

==	uguale (da non confondere con = !!!!!!!!!)
<	minore
<=	minore o uguale
>	maggiore
>=	maggiore o uguale
!=	non uguale

- Operatori logici:

!	NOT (operatore unario)
&&	AND (operatore binario)
	OR (operatore binario)

- Restituiscono tutti un numero: 0 per “falso”, >0 per “vero”
- Ad esempio, OR restituisce un valore > 0 se uno degli operandi è >0, restituisce 0 se entrambi gli operandi sono nulli

Esempio

```
int x=5, y, z;  
y = (x==5);           \\ y=1 (o comunque un valore >0)  
z = (x=5);            \\ z=5  
z = (x>5);            \\ z=0  
x = (x=x);            \\ x inalterato  
x = (x==x);           \\ x=1 (o comunque un valore >0)  
x = (x!=x);           \\ x=0  
z = ((x<y) && (y!=1))  \\ z=0  
z = ((x<y) || (y!=1))  \\ z=1 (o comunque un valore >0)  
z = !(x<y)            \\ z=0
```

NB: studieremo gli operatori AND, OR, NOT con l'algebra booleana

Costanti

- Rappresentano associazioni identificatore - costante:
nel programma utilizzo l'identificatore al posto del valore associato
- La loro definizione è simile a quella delle variabili, premettendo la parola chiave `const`
- Esempio:

```
const char acapo = '\n';  
const float nofebbre = 36.6;  
...  
if(temperatura>nofebbre)  
    printf("Hai la febbre\n");
```


Commenti in C

- Si possono inserire commenti per spiegare alcuni dettagli del codice: ovviamente è necessario che vengano “marcati” affinché il compilatore li possa ignorare
- Tutto ciò che è tra `/*` e `*/` (anche su più righe) è un commento:

```
/* questo è  
un commento */
```

- Quasi tutti i compilatori ammettono anche i commenti nella forma

```
printf("Ciao\n"); //anche questo è un commento
```

tutto ciò che segue `//` è un commento fino alla fine della riga
(in questo caso quindi il commento non è su più righe)

Nota sul tipo char

- I caratteri vengono rappresentati come “piccoli numeri interi” (compresi tra 0 e 255)
- In un’operazione di assegnamento, ad esempio
 char carattere;
 carattere = 'a';
la variabile assume valore intero corrispondente al codice ASCII, nell’esempio carattere risulta pari a 97
- Si possono utilizzare operatori relazionali, ad esempio
 'a' < 'b' risulta vero
(il codice ASCII mantiene l’ordinamento alfabetico)
- La funzione di stampa che useremo consente poi di specificare il formato di stampa (come carattere o come intero, vedi poi)

Funzione di stampa printf

Effetto:

- stampa in uscita la stringa di controllo, che contiene caratteri comuni e *caratteri di conversione* preceduti dal simbolo % (ad esempio: %d, %f, %c)
- ad ogni carattere di conversione corrisponde un elemento da stampare nella lista
- un caratteri di conversione provoca la stampa del corrispondente elemento, dopo averlo convertito nel formato di stampa specificato dal carattere di conversione stesso (es: %d per il formato decimale)

Una sequenza speciale

`\n` rappresenta il carattere di “a capo”

Es.

```
printf("fuori");  
printf("gioco\n");
```

} Stampa “fuorigioco” e va a capo

Caratteri di conversione

`%d` notazione decimale

`%o` notazione ottale

`%x` notazione esadecimale

`%c` si considera l'argomento un carattere

`%s` si considera l'argomento una stringa (vedremo in seguito)

`%f` notazione in virgola fissa (per valori reali)

`%e` notazione in virgola mobile

`%g` notazione più semplice tra virgola fissa e virgola mobile

`.` specifica quante cifre dopo la virgola

Esempi

```
int g1=1;
```

```
int g2=0;
```

```
printf("Milan %d Inter %d, %s",g1,g2,"Ronaldo");
```

Stampa “Milan 1 Inter 0, Ronaldo”

```
float gradi=36.6;
```

```
printf("La tua temperatura è di %f gradi", gradi);
```

Stampa “La tua temperatura è di 36.6 gradi”

```
char carattere='a';
```

```
printf("Il carattere è %c, ma anche %d\n", carattere,carattere);
```

Stampa “Il carattere è a, ma anche 97”

Funzione di acquisizione dati scanf

scanf(stringa di controllo, elementi da acquisire);

tra apici contiene solo caratteri di conversione	lista di variabili separate dalla virgola, ciascuna preceduta dal simbolo &
--	---

Es.

```
scanf("%d", &valore);
```

- La funzione scanf legge caratteri dalla tastiera e li interpreta secondo il formato specificato dai caratteri di conversione
- Ciascun valore così ottenuto è immagazzinato in un argomento della lista
- Gli argomenti della lista indicano in realtà indirizzi di memoria in cui memorizzare i valori acquisiti (per questo si usa &)

Caratteri di conversione

%d	si aspetta di acquisire un intero decimale
%o	si aspetta di acquisire un intero ottale
%x	si aspetta di acquisire un intero esadecimale
%h	si aspetta di acquisire un intero short
%c	si aspetta di acquisire un carattere
%s	si aspetta di acquisire una stringa (vedremo in seguito)
%f	si aspetta di acquisire un valore reale

Esempio

```
#include<stdio.h>
#include <stdlib.h>

main(){
    int primo, secondo, somma;

    printf("Inserisci due numeri interi\n");
    scanf("%d%d",&primo,&secondo);
    somma=primo+secondo;
    printf("Somma uguale a %d\n",somma);

    system("pause");
}
```


Un altro esempio

```
#include<stdio.h>
#include <stdlib.h>

main(){
    int primo, secondo;
    int somma;

    printf("Inserisci il primo numero\n");
    scanf("%d",&primo);
    printf("Inserisci il secondo numero\n");
    scanf("%d",&secondo);
    somma=primo+secondo;
    printf("Somma uguale a %d\n",somma);
    system("pause");
}
```

Funzione system

- Fa parte della libreria *stdlib* (ecco perché abbiamo incluso `stdlib.h`)
- E' una funzione che esegue comandi del sistema operativo
- Ad esempio, la chiamata `system("pause")` causa l'esecuzione del programma `pause`
- Questo programma stampa a video
 “Premere un tasto per continuare”
e aspetta che l'utente digiti un tasto prima di continuare
- Noi lo utilizziamo per fare in modo che la finestra di esecuzione non scompaia prima che l'utente abbia potuto vedere il risultato dell'esecuzione del programma eseguibile!