# **Guida breve alla programmazione in C:**
Prontuario delle istruzioni più usate

*Giulio Del Corso*

**Indice:**

**Primi passi:**

**Programma per scrivere:**
Linux: Gedit
Windows: Dev C++

Il formato del file su cui scrivere è:
nomefile.c

Un compilatore è gcc:
gcc nome_file.c -o nome_eseguibile

Il comando per far partire un eseguibile è:
./nome_eseguibile

**Commenti:**
/* Commento */

**Attenzione:**
Ogni istruzione in C deve terminare con ;
Le maiuscole sono considerate diverse dalle minuscole.

**Librerie:**

All'inizio di un programma dobbiamo aggiungere le librerie che intendiamo utilizzare:

#include<stdio.h>
Standard input – output

#include<stdlib.h>
Libreria di funzioni predefine e allocazione dinamica.

#include<math.h>
Libreria di funzioni matematiche più avanzate.

**Attenzione:** compilare con l'istruzione:
gcc -lm nome_file.c -o nome_eseguibile

**Osservazione:**
Per compilatori aggiornati l'istruzione -lm non è necessaria.

**Funzioni contenute nella libreria math:**

sqrt(x)
Radice di x
abs(x)
Valore assoluto di x
log(x)
Logaritmo naturale
log10(x)
Logaritmo in base 10
pow(x,y)
Potenza x^y
floor(x)
Intero minore di x
ceil(x)
Intero maggiore di x
sin(x)
Seno di x
cos(x)
Coseno di x
tan(x)
Tangente di x
sinh(x)
Seno iperbolico di x
cosh(x)
Coseno iperbolico di x
tanh(x)
Tangente iperobolica di x

c=a%b (Interi) restituisce il resto della divisione fra a e b. (Presente senza la libreria math.h)

**Dichiarazione di variabili globali:**

Vengono definite fra le librerie e il main; sono variabili che rimangono valide sia nel main che nelle funzioni/procedure successive.

> **Osservazione importante:**
> Dichiarando una variabile globale ogni funzione/procedura ha attribuiti i puntatori a quella variabile.
> Di conseguanza variando quella variabile in una qualsiasi procedura questa risulterà modificata anche nel main.
> Questo permette di lavorare senza passare esplicitamente i puntatori.

**Parte centrale del programma:**

```
main()
    {
    Parte dichiarativa
    Corpo del programma
    }
```

**Parametro in fase di compilazione:**

```
#include<stdio.h>

#define Iterazioni 10
```

Sostituisce in fase di compilazione “Iterazioni” con il valore “10”.

**Tipi strutturati:**

Si dichiarano prima del main.

*Esempio:*

```
struct nome_struttura
        {
        int var1 ;
        int var2 ;
        int vet1[10] ;
        } ;
```

**Osservazione:**
Quando dichiariamo il tipo (nel main o nelle subroutines) strutturato dobbiamo inserire la parola struct.

```
struct nome_struttura nome_variabile ;
```

**Osservazione:**
Si può assegnare una variabile strutturata ad un'altra dello stesso tipo.

```
struct nome_struttura nome_variabile1 , nome_variabile2 ;
nome_variabile1=nome_variabile2 ;
```

**Attenzione:**
Non si possono confrontare:

```
nome_variabile1==nome_variabile2
```

**Accedere ai campi della struttura:**
Si utilizza l'operatore punto:

```
nome_variabile.var1
```

oppure:

```
nome_variabile.vet1[posto]
```

**Utilizzando i puntatori:**

```
(*puntatore).var1
```

Nel caso di puntatori questo è equivalente a:

```
puntatore->var1
```

**Osservazione:**
Possono essere inizializzate per elencazione:

```
struct nome_struttura nome_variabile={4,7,{1,3}} ;
```

I valori non specificati vengono azzerati.

**Liste:**
Sono usate per allocare dinamicamente lo spazio.
Una lista è un tipo strutturato che contiene informazioni e un puntatore all'elemento successivo della lista.

```
struct EL
        {
        int valore ;
        struct EL *next ;
        } ;

typedef struct EL ElementoLista ;
typedef ElementoLista *ListaElementi ;
```

**Typedef:**
Questo comando permette di rinominare un tipo (strutturato) o un puntatore ad esso per accedervi con maggiore facilità.

Sintassi:

```
typedef struct EL ElementoLista ;
```

A questo punto nel main definire il tipo ElementoLista è equivalente a struct EL.

Si può in maniera analoga definire un tipo puntatore:

```
typedef ElementoLista *ListaElementi ;
```

**Attenzione:**
ListaElementi è un puntatore.

**Osservazione ultimo elemento:**
L'ultimo elemento deve avere next che punta a NULL così da terminare la lista.

**Creare un tipo definito dall'utente:**

typedef int anno ;

anno è diventato un tipo intero che può definire altre variabili.

Nel main possiamo scrivere:
anno a=2012 ;

**Osservazione:**
Si scrive nella fase dichiarativa globale.

**Osservazione:**
Possiamo definirle per enumerazione:

typedef enum{lun,mar,mer,gio,ven,sab,dom} giorni

**Attenzione:**
Di base C considera questi elementi di tipo intero, possiamo quindi confrontarli o fare lun+mar.
Non possiamo però stamparli (otterremmo degli interi), un sistema potrebbe essere utilizzare uno switch.

**Allocazione dinamica:**
Si utilizzano i comandi malloc e free.

Definito un tipo e una variabile puntatore a quel tipo si può assegnare al puntatore l'indirizzo mediante il comando:

```
p=malloc(sizeof(tipo)) ;
```

A questo punto possiamo creare una lista di elementi caratterizzati esclusivamente dal puntatore ad essi (sono anonimi).

**Attenzione:**
Perdere il puntatore alla lista significa perdere i valori in essa contenuti.

**Osservazione:**
Per generare una lista ogni passo deve essere allocato mediante malloc.

free libera lo spazio allocato con malloc:

```
free(p) ;
```

**Attenzione:**
Non rende NULL il puntatore, si limita a liberare la memoria.

**Osservazione:**
E' buona norma usare free per ridurre la generazione di memoria allocata senza puntatori ad essa.

**Funzioni:**

int nome_funzione(int nome_variabile_input_1, int  nome_variabile_input_2)
{
Dichiarazione variabili
Corpo funzione
}

**Osservazione:**
Nella dichiarazione degli input dobbiamo dichiarare il tipo.

**Osservazione:**
Per modificare oggetti multipli dobbiamo passargli gli indirizzi, altrimenti si limita a restituire un valore (o un tipo strutturato).

**Comando di chiusura:**
return nome_variabile ;
Chiude la funzione restituendo il valore.

**Procedure:**
Sono funzioni di tipo void, non restituiscono un valore.

*Esempio utilizzo:*
Operazioni su di un vettore, stampa di un vettore.

Void nome_procedura(int var1, int var2)
{
Dichiarazione variabili
Corpo procedura
}

**Osservazione:**
return nel caso di procedure è equivalente a break.

**Chiamare una funzione:**
Nel corpo della funzione main utilizziamo il comando:
a=nome_funzione(variabile1,variabile2) ;

**Chiamare una procedura:**
Nel corpo della funzione main utilizziamo il comando:
nome_procedura(variabile1,variabile2) ;

**Definire all'inizio le funzioni/procedure:**
Le funzioni e le procedure possono essere definite all'inizio usando la notazione:

```
void nome_procedura(int, float) ;
int nome_funzione(int) ;
…
```

**Osservazione:**
Questo garantisce, nel caso di procedure, che il compilatore riconosca nome_procedura come una procedura e non come una variabile.

**Attenzione:**
Si dichiara il tipo ma non il nome della variabile passata.

**Osservazione:**
Le funzioni possono essere dichiarate all'inizio per una questione di comodità di programmazione.
Questo però non è necessario e non è importante l'ordine in cui le scriviamo all'interno del file in quanto il compilatore individua quelle chiamate.

**Corpo di una funzione, parte dichiarativa:**

Nella parte dichiarativa si vanno a definire le variabili locali del programma.

Ogni variabile deve aver assegnato un tipo seguendo la notazione:

```
tipovariabile nomevariabile ;
```

**Tipi variabili:**

int

Numeri interi.
In ordine crescente di bit utilizzati: short, int, long.
Aggiungere unsigned rende gli interi privi di segno.

```
unsigned int a ;
```

float

Numero reale.
In ordine crescente di dimensione: float, double, long double

Come formati di printf e scanf si usano:

| | float | double | long double |
|---|---|---|---|
| printf | %f | %f | %Lf |
| scanf | %f | %lf | %Lf |

char

Caratteri (8 bit).
Per assegnare un carattere si utilizza l'apice semplice.

```
char a = 'B' ;
```

**Inizializzazione:**

Ogni variabile può essere inizializzata sfruttando il comando:

```
int a=3 ;
```

**Costanti:**

Aggiungere il comando const impedisce la modifica della variabile nel corso della parte esecutiva:

```
const pi=3.14 ;
```

**Dichiarazione multipla:**

```
int a , b , c ;
```

**Corpo di una funzione, parte esecutiva:**

**Comandi di output grafico:**
printf(“Testo da stampare su schermo.”)

\n Inserisce un rientro a capo.

Per inserire dei valori nella stringa stampata si usa il comando:
printf(“Inserire %d”,valore)

**Formato da inserire:**
%d Stampa un intero in notazione decimale (Tipo intero)
%hd per il tipo short.
%ld per il tipo long.
In generale per il tipo short si antempone h, per il tipo long si antepone l.

%u Stampa un intero di tipo unsigned

%c Stampa un carattere.
**Attenzione:** stampa un carattere singolo, non una stringa.

%s Stampa una stringa.

%p Stampa un puntatore, quindi un indirizzo ad una variabile

**Assegnamento:**
a = 3 ;

**Modifica implicita dei tipi:**
a = b
b viene convertito al tipo di a

(a+b)
Viene calcolato nel tipo di livello più alto, per un int e un float diventa float.
I caratteri in queste operazioni vengoni convertiti in int.

**Stampa di un carattere memorizzato in c:**
putchar(c)

**Acquisizione di un carattere per la variabile c:**
c=getchar()

**Input da tastiera:**

scanf(“%d”, &variabile)

Con %d il formato della variabile e &variabile l'indirizzo della variabile.

**Formato da inserire:**
%d Legge un intero in notazione decimale (Tipo intero)
%hd per il tipo short.
%ld per il tipo long.
In generale per il tipo short si antempone h, per il tipo long si antepone l.

%u Legge un intero di tipo unsigned

%c Legge un carattere.
scanf(“%c %c”,&a,&b) legge due caratteri intervallati da uno spazio.
%* fa saltare un input.
scanf(“%c%*%c”,&a,&b) legge due caratteri intervallati da un carattere qualsiasi.

%s Legge una stringa.

Aspetta un invio per confermare la lettura.

**Operatori logici:**
== Uguale
< Minore
<= Minore uguale
> Maggiore
>= Maggiore uguale

! Diverso
!= Diverso da.

&& Congiunzione

|| Disgiunzione

**Osservazione:**
(a==b) è un valore intero pari a 0 se la proposizione è falsa, 1 se è vera.

**Costrutti iterativi e condizionali:**

**If:**
*Sintassi:*

```
if (condizione logica)
        {
        Istruzioni
        }

if (condizione logica)
        Istruzione singola

if (condizione logica)
        {
        Istruzioni
        }
else
        {
        Istruzioni
        }
```

**Else annidati:**

```
if (condizione logica)
        Istruzione 1
else if (condizione logica)
        Istruzione 2
else if (condizione logica)
        Istruzione 3
else
        Istruzione alternativa a tutte le precedenti
```

**Ciclo While:**

while (espressione logica)
{
Istruzioni
}

while (espressione logica)
Istruzione singola

**Ciclo do while:**
do
{
Istruzioni
}
while (condizione logica) ;

**Osservazione:**
Esegue le operazioni prima del controllo logico.

**Ciclo for:**

for (i=valore_iniziale ; i<=condizione ; i=i+1)
    Istruzione singola

    **Attenzione:**
    i deve essere dichiarata di tipo intero.
    La condizione non può essere un'uguaglianza.

    **Osservazione:**
    Il contatore può essere modificato all'interno del ciclo for.

**Comandi utili:**
i++ è equivalente i=i+1
i-- è equivalente a i=i-1

**I vettori:**

**Dichiarazione:**
int nome_vettore[6]

Crea un vettore di 6 elementi di tipo intero.

**Attenzione:**
Il vettore è numerato a partire da 0

L'elemento i-esimo si richiama con il comando:
vet[i]

**Allocazione:**
int vet[4]={1,2,5,12} ;
Se vengono assegnati meno elementi della dimensione del vettore gli altri vengono posti uguali a 0.

**Attenzione:**
C non accetta comandi di tipo vettoriale.

**Vettori multidimensionali:**

**Dichiarazione:**
int nome_vet[6][4] ;
Matrice di 6 righe e 4 colonne.

Si richiamano inserendo le coordinate (Ricordando che partono entrambe da 0).

**Inizializzazione:**
Inserisce i vettori per righe successive, nel caso multidimensionale riempie prima la prima variabile, poi la seconda, etc.

*Esempio (3):*
(100;200;300;010;110;210;310,020;...;001,101;201;301;011;...)

**Puntatori:**

Possiamo dichiare i puntatori ad un tipo mediante la dichiarazione:

int *pi

pi è l'indirizzo di una variabile intera.

**Osservazione 1:**

&a

Indirizzo di una variabile.

**Attenzione:**
a è una variabile.

Si può attribuire un indirizzo ad un puntatore:
pi=&a

Si dice che pi punta alla variabile intera a.

**Osservazione 2:**

*pi

Valore contenuto all'indirizzo pi, equivalente valore puntato da pi.

**Attenzione:**
pi è un puntatore.

Può essere usato per modificare il valore di ciò che è puntato:

```
*pi=17 ;
*pi=*pi+3 ;
```

**Osservazione stampa:**

Si possono usare i puntatori per stampare i valori da essi puntati:

```
printf(“Valore: %d”,*pi) ;
```

**Osservazione confronti logici:**

(p==q) è 1 se l'indirizzo dei due puntatori coincide.
(*p==*q) è 1 se il valore puntato dai due puntatori coincide.

**Puntatore a puntatore:**

**Dichiarazione:**
int **ppi , *pi , i ;
ppi=&(pi) ;
Il puntatore ppi punta all'indirizzo del puntatore pi.
pi=&i
Il puntatore pi punta all'indirizzo dell'intero i.

Quindi *ppi è il contenuto di ciò che è puntato da ppi, quindi è l'indirizzo di i.
Inoltre **ppi è proprio il valore di i.

**Osservazione:**
Possiamo fare puntatori ai puntatori di puntatori di interi mediante notazione:
int ***ppi , **ppi , *pi , i ;
etc.

**Osservazione:**
In quanto puntatori di puntatori si stampano gli indirizzi loro (o dei loro puntati) sempre con il formato %p.

**Vettori e puntatori:**
Quando definiamo vet[3] in realtà abbiamo assegnato l'indirizzo vet del primo elemento del vettore.

Quindi:
vet[2]=*(vet+2) (Parte esecutiva)

Possiamo quindi assegnare ad un puntatore il valore (indirizzo) vet.
int *pi , vet[3] ;
pi=vet ;

**Equivalente:**
int vet[3] ;
int *pi=vet ;

**Osservazione:**
*(pi+3) è uguale a vet[3]
pi+3 è uguale a &vet[3]

**Attenzione:**
Possiamo modificare il valore del puntatore pi ma non del puntatore vet.

**Passare un puntatore ad una funzione:**
Può essere usato per modificare i valori di una variabile.

**Sintassi:**
```
nome_funzione(&a) ;
….
void nome_funzione(int *pi)
        {
        Istruzioni
        }
```

**Osservazione:**
Nella funzione che sfrutta il puntatore (*pi) possiamo modificare il valore da esso puntato lavorando con *pi.

**Osservazione:**
Nella fase di dichiarazione iniziale il tipo lo definiamo come:

```
void nome_funzione(int*) ;
```

**Funzioni ricorsive:**

Una funzione può chiamare se stessa in maniera tale da renderla ricorsiva.

*Esempio fattoriale:*

```
int funzione(int n)
        {
        if (n==0)
                return 1 ;
        else
                return n*funzione(n-1) ;
        }
```

**Osservazione:**
Possiamo nel return richiamare anche più di una volta la funzione:

```
return funzione(n-1)+n*funzione(n-3) ;
```

**Attenzione:**
Bisogna controllare che esista una condizione di terminazione, se non garantita dobbiamo imporre un caso base con if per il return.

**Lavorare con un file di testo:**

Bisogna osservare che la funzione di chiamata di un file restituisce un puntatore a file.

In fase dichiarativa dobbiamo dunque porre:

```
FILE *fd ;
```

**Comando di apertura:**

```
fd=fopen("nome_file.txt","modo_apertura") ;
```

**Modi di apertura:**

r (read)

Impedisce la modifica del file, non crea un file se il nome del file è sbagliato.

w (write)

Scrive sul file, lo genera se non esiste un file con quel nome. Prestare attenzione al fatto che se il file già esiste lo sovrascrive.

a (append)

Scrive sul file senza sovrascriverlo, tutto ciò che aggiunge lo mette dopo quanto già scritto.

**Controllo:**

Se il file non è stato aperto correttamente il puntatore fd ha valore NULL.

**Comando di chiusura:**

```
fclose(puntatore_file) ;
```

*Esempio:*

```
fclose(fd) ;
```

**Osservazione:**

Il file a quel punto non è più accessibile ma i dati memorizzati sulle variabili della funzione si.

E' buona norma chiudere un file una volta utilizzato.

**Leggere da un file:**
Si utilizza il comando:

fscanf(puntatore_file,"%d",&var) ;

**Osservazione:**
Se usato mediante un ciclo for legge i valori successivi.

**Attenzione:**
I valori nel file devono essere separati da uno spazio per essere leggibili, di conseguenza però la formulazione corretta di fscanf deve essere con lo spazio:

fscanf(puntatore_file,"%d ",&var) ;

**Scrivere su di un file:**
Si utilizza il comando:

fprintf(puntatore_file,"%d ",var) ;