
Puntatori a funzione.

In C è possibile utilizzare dei puntatori a funzioni, ovvero delle variabili a cui possono essere assegnati gli indirizzi in cui risiedono le funzioni, e tramite questi puntatori a funzione, le funzioni puntate possono essere chiamate all'esecuzione.

Confrontiamo la **dichiarazione di una funzione**:

tipo_restituito **nome_funzione** (paramdef1, paramdef2, ...)

con la **dichiarazione di un puntatore a funzione**:

tipo_restituito **(* nome_ptr_a_funz)** (paramdef1, paramdef2, ...)

dove:

- paramdef1, paramdef2, ecc. sono la definizione degli argomenti da passare alla funzione all'atto della chiamata (ad es.: int i).
- tipo_restituito è il tipo del dato che viene restituito come risultato dalla funzione.

ad es, la seguente dichiarazione definisce un puntatore a funzione che punta a funzioni le quali prendono come argomenti due double, e restituiscono un double (void).

double (***ptrf**) (double g, double f);

Il C tratta i nomi delle funzioni come se fossero dei puntatori alle funzioni stesse.

Quindi, **quando vogliamo assegnare ad un puntatore a funzione l'indirizzo di una certa funzione dobbiamo effettuare un operazione di assegnamento del nome della funzione al nome del puntatore a funzione**

Se ad es. consideriamo la funzione dell'esempio precedente:

double somma(double a, double b);

allora **potremo assegnare la funzione somma al puntatore ptrf** così:

ptrf = somma;

Analogamente, **l'esecuzione di una funzione mediante un puntatore che la punta,viene effettuata con una chiamata in cui compare il nome del puntatore come se fosse il nome della funzione**, seguito ovviamente dai necessari parametri.

---------- esempio di uso dei puntatori a funzione ----------

per es. riprendiamo l'esempio della somma:

```
double somma( double a, double b) ;          /*  dichiarazione  */

void main( void)
{
      double A=10 , B=29,  C;
      double (*ptrf) ( double g, double f);

      ptrf = somma;
      C = ptrf (A,B);                        /* chiamata alla funz. somma */
}

double somma( double a, double b)            /* definizione */
{     return a+b ;     }
```

Osservazione: spesso è complicato definire il tipo di dato puntatori a funzione, ed ancora di più definire funzioni che prendono in input argomenti di tipo puntatori a funzione.
In queste situazioni **è sempre bene ricorrere alla typedef per creare un tipo di dato puntatore a funzione** per funzioni che ci servono, ed utilizzare questo tipo di dato nelle altre definizioni.

---- Usare la typedef per rendere leggibile il codice C ---

Esempio:
esiste in ambiente unix (l'esempio è preso da LINUX Slackware) una funzione detta signal che puo' servire ad associare una funzione al verificarsi di un evento. Ad es. puo' servire a far eseguire una funzione allo scadere di un timer.
Il prototipo della funzione, contenuto in signal.h, e' il seguente:

```
#include <signal.h>
void (*signal(int signum, void (*handler)(int) ) )(int);          /* ????? */
```

NON E' SUBITO CHIARISSIMO COSA SIA STA ROBA !!!

Il significato è che
1) la funzione signal vuole come parametri un intero ***signum***, ed un puntatore ***handler*** ad una funzione che restituisce un void e che vuole come parametro un intero.
2) la funzione signal restituisce un puntatore ad una funzione che restituisce un void e che vuole come parametro un intero.

Converrebbe definire un tipo di dato come il puntatore a funzione richiesta:

```
typedef  void (*tipo_funzione) (int);
```

ed utilizzarlo per definire la signal così:

```
tipo_funzione   signal ( int signum,  tipo_funzione handler );
```

Nel man della signal e' presente questo commento:

> If you're confused by the prototype at the top of this manpage, it may help to see it separated out thus:
>
> ```
> typedef void (*handler_type)(int);
> handler_type signal(int signum, handler_type handler);
> ```

INPUT ed OUTPUT

Le funzioni della libreria standard per il C per l'I/O sono definite nell'header file **stdio.h** . (input / output standard)

Quando un programma C entra in esecuzione l'ambiente del sistema operativo si incarica di aprire 3 files di I/O, ovvero 3 flussi di dati, e di fornire i rispettivi puntatori di tipo FILE * globali.

La struttura chiamata FILE contiene le informazioni fondamentali sul file, quali il modo di apertura del file, il nome del file, l'indirizzo di un buffer in cui sono conservati i dati letti dal file e la posizione corrente nel buffer. L'utente non deve conoscere questi dettagli, ma deve memorizzare in una variabile di tipo **puntatore a FILE** (FILE*) l'indirizzo di questa struttura per utilizzarla nelle letture o scritture su file.

I flussi di dati standard sono:
STANDARD INPUT serve per l'input normale (per default da tastiera), e ha come puntatore a FILE la variabile globale **stdin**. Dallo standatd input prendono i dati le funzioni getchar, gets e scanf.
STANDARD OUTPUT serve per l'output normale (per default su schermo), e ha come puntatore a FILE la variabile globale **stdout**. Sullo standatd output scrivono i loro dati le funzioni putc, printf e puts
STANDARD ERROR serve per l'output che serve a comunicare messaggi di errore all'utente (per default anche questo su schermo), e ha come puntatore a FILE la variabile globale **stderr**.

Stdin, stdout e stderr sono delle costanti globali contenute in stdio.h,. e non possono essere modificate.
Questi flussi possono pero' essere ridirezionati in modo da scrivere su file su disco, oppure di leggere da file su disco.

INPUT

- Il sistema prevede che **l'input sia organizzato a linee, ciascuna terminata da un carattere new line (ENTER \n),** solitamente fornite dall'utente tramite la tastiera, ma a volte anche fornite via file.
- L'utente da tastiera puo' digitare i caratteri e cancellarli, ma **tali caratteri non vengono passati all'applicazione fino a che l'utente non preme <RETURN>** nel qual caso i dati presenti sul buffer di tastiera vengono inseriti in un vettore di caratteri, **e in fondo viene aggiunto un carattere NEW LINE** (individuato da \n).
- NOTARE che **l'utente non ha a disposizione i caratteri fino a che non viene premuto RETURN**, dopodiche i dati, in forma di linea di testo terminante con un NEW_LINE sono pronti per essere letti dall'applicazione.

La lettura dei dati puo' essere effettuata **dallo standard input un carattere alla volta oppure una linea alla volta.**

La lettura carattere per carattere viene effettuata mediante la funzione:

int getchar(void);

che restituisce il successivo carattere in input in forma di codice ASCII, cioe' restituisce ad es. 65 per 'A', 66 per 'B', ecc. 97 per 'a'. 98 per 'b', oppure restituisce EOF quando incontra la fine del flusso di input, che viene rappresentata come la fine del file di input.

Tale funzione e' bloccante nel senso che quando una linea di caratteri precedentemente data in input e' finita, rimane in attesa che dallo standard input arrivi una nuova linea di dati.

La lettura di una intera linea viene effettuata mediante la funzione:

char *gets(char *dest**);**

che scrive in dest la linea e restituisce un puntatore a dest se tutto va bene, restituisce NULL altrimenti.