
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);
> ```

Funzioni con numero di argomenti variabile.

In C e' possibile definire funzioni aventi un numero di argomenti variabile, cioe' funzioni che in chiamate diverse possono avere un numero di argomenti diverso, **ma ne debbono avere sempre almeno uno**. Ne è un esempio la printf();
Si utilizza a questo scopo una struttura **va_list** definita nel file stdarg.h .
Vediamo un esempio di funzione che riceve in input n argomenti, di cui il primo e' un intero che contiene il numero di argomenti seguenti, e gli altri sono delle stringhe, cioe' dei puntatori a char. La funzione deve solo stampare tutti gli argomenti passati.

```
void print_lista_argomenti(int narg, ...)
{
va_list va;
int i;
char *ptrchar;

/* va_start inizializza va_list all'argomento, tra passati a print_lista_argomenti, che segue l'argomento narg indicato come secondo argomento nella va_start, cioè inizializza la lista va_list al primo degli argomenti variabili  */
va_start(va,narg);
for(i=0;i<narg;i++)
   {
   ptrchar=va_arg(va,char*);  /*ptrchar punta alla stringa passata*/
   printf("%d %s\n",i,ptrchar);
   }
va_end(va);
}
```

la funzione potra' essere chiamata in una di questi modi

```
print_lista_argomenti(0);
print_lista_argomenti(5,"primo","secondo","terzo","quarto","quinto");
```

Vediamo un esempio complicato ma utile di uso della lista di argomenti variabili. Stampa gli argomenti passati, anche se sono di tipo diverso. Usa un primo parametro come formato per sapere cosa viene passato nei successivi argomenti, indicando con s una stringa, con d un intero, con c un carattere.

```
#include <stdio.h>
#include <stdarg.h>
void stampa_argomenti(char *fmt, ...)
{
    va_list ap;
    int d;
    char c, *p, *s;

    va_start(ap, fmt);
    while (*fmt)
        switch(*fmt++) {
        case 's':          /* string */
            s = va_arg(ap, char *);
            printf("string %s\n", s);
            break;
        case 'd':          /* int */
            d = va_arg(ap, int);
            printf("int %d\n", d);
            break;
        case 'c':          /* char */
            c = va_arg(ap, char);
            printf("char %c\n", c);
            break;
        }
    va_end(ap);
}

void main (void) {
      stampa_argomenti ( "sdc" , "stringa" , (int) 10 , (char)'h' );
      stampa_argomenti ("s" , "stringa2");
}
```

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.

Fine dell'INPUT

Se l'utente preme contemporaneamente la coppia di tasti (**CTRL+d in UNIX**, **CTRL+z in DOS** seguito prima o poi da RETURN), in un certo senso segnala all'applicazione che il flusso di dati e' terminato.

Nella linea di testo viene aggiunto un carattere EOF (che di solito e' rappresentato dal valore -1) ma che e' bene sempre confrontare con la costante EOF (definita in stdio.h), ed il flusso di input viene chiuso, cioe' da quel momento in avanti ogni successiva chiamata a getchar restituira' sempre EOF.

ESEMPIO di lettura dallo standard input

```
#include <stdio.h>
void main(void)
{
int ch;
while(  (ch=getchar()) != EOF )
        {
        /* uso il carattere, ad es lo incremento di 1 e lo stampo*/
        ch ++;
        putchar(ch);
        }
}
```

RIDIREZIONAMENTO DI INPUT ED OUTPUT

L'utente puo' utilizzare lo standard input dare input non solo da tastiera ma anche da file, ridirezionando un file sullo standard input nel seguente modo, al momento dell'esecuzione del programma:

```
program < file_input
```

Analogamente l'output di un programma puo' essere ridirezionato su un file, su cui sara' scritto tutto l'output del programma invece che su video

```
program > file_output
```

e le due cose possono essere fatte assieme

```
program < file_input > file_output
```

In UNIX si può ridirezionare assieme standard input e standard error:

```
program  >&  file_error_and_output
```

OUTPUT CARATTERE PER CARATTERE (NON FORMATTATO) su stdout.

La scrittura dei dati sullo standard output (video) puo' essere effettuata **un carattere alla volta** mediante la funzione:

int putchar(int ch);

che restituisce il carattere scritto ch se va tutto bene, altrimenti restituisce EOF in caso di errore.

OUTPUT FORMATTATO su stdout

L'output puo' essere effettuato anche in maniera formattata, e non solo un carattere alla volta.

La funzione printf serve proprio a stampare un output a blocchi sullo standard output secondo un formato specificato, traducendo le variabili in caratteri, cioe' printf stampa sullo standard output gli argomenti arg1, arg2, ... secondo le specifiche contenute nel format.

int printf(char *format, ...);

meno formalmente

int printf(char *format, arg1, arg2, arg 3, ...);

il format e' una stringa di caratteri null-terminata, in cui compaiono o dei semplici caratteri che verranno stampati cosi' come sono oppure degli specificatori di formato nella forma **%qualcosa**.

"%d"	stampa un intero
"%10d"	stampa un intero e un minimo di 10 caratteri almeno in cui l'intero e' a destra
"%-10d"	stampa un intero e un minimo di 10 caratteri almeno in cui l'intero e' a sinistra
"%f"	stampa un float
"%lf"	stampa un double
"%10f"	stampa un float e un minimo di 10 caratteri almeno in cui il float e' a destra
"%-10f"	stampa un float e un minimo di 10 caratteri almeno in cui il float e' a sinistra

"%10.5f"	stampa un float e un minimo di 10 caratteri con al massimo 5 cifre dopo il punto decimale
"%s"	stampa tutti i caratteri di una stringa (cioe' un vettore di caratteri con un \0 in fondo) fino a che incontra lo 0 finale
"%10s"	stampa almeno 10 caratteri di una stringa, con la stringa a destra
"%15.10s"	stampa almeno 15 caratteri, prendendone al massimo 10 dalla stringa, con la stringa a destra
"%c"	stampa un singolo carattere

NB. per stampare un % devo scrivere %%

NB alcuni caratteri di controllo possono essere scritti mediante la loro sequenza di escape

NEW LINE (a capo linea)	\n	
TABULAZIONE	\t	
BACKSPACE	\b	torna indietro di un carattere ma non lo cancella se pero' scrivo qualcosa gli passo sopra

NB poiché le stringhe vengono delimitate da due doppi apici " per stampare un doppio apice devo scriverlo preceduto da un BACKSLASH cioè devo scrivere \"
Analogamente, poiché la \ serve a indicare una sequenza di escape, per stampare una \ devo scriverne due. (vedi nomi file nella fopen).

esempio:

```
int i=16;  double g=107.13987626; char *str="pippo";
      printf("%d)  %10.3lf \"%s\"\n",i,g,str);
```

da' come risultato

```
      16)     107.139 "pippo"
```

INPUT FORMATTATO da stdin (1)

Per estrarre dal flusso dello standard input i dati e' spesso utilizzata la funzione scanf. La scanf cioe' prende l'input dallo standard input (aspetta un RETURN) e cerca di estrarre i dati che sono specificati nel suo primo agomento

int scanf(char *format, ...);
int scanf(char *format, &arg1, &arg2, &arg 3, ...);

I dati estratti vengono scritti nelle variabili i cui indirizzi sono passati come argomenti successivi al primo nella scanf. NB. L'errore classico e' passare come argomento la variabile e non il suo indirizzo.

La stringa di formato puo' contenere sia dei semplici caratteri, che allora devono corrispondere ai caratteri che vengono digitati in input, sia degli **specificatori di conversione** nella forma **%qualcosa** che indicano come devono essere interpretati i caratteri che costituiscono il flusso di input.

%c	viene letto un singolo carattere e copiato singolarmente nel primo byte indicato dal puntatore
%10c	vengono letti 10 caratteri e copiati nei primi 10 byte indicati dal puntatore
%s	viene letta la stringa e scritta interamente a partire dal puntatore passato, e in fondo viene aggiunto un carattere '\0' di terminazione
%d	viene letto un intero
%f	viene letto un float
%lf	viene letto un double

La **scanf termina** quando esaurisce la sua stringa di formato o quando verifica una inconsistenza tra l'input e le specifiche del formato.
La **scanf restituisce** il numero di elementi trovati. Se non è stato scritto nessun elemento possono esserci due motivi diversi: o lo standard input era stato chiuso e allora la scanf restituisce EOF, oppure lo standard input era ancora aperto ma l'input non era consistente con le richieste del formato e c'è stato errore nella conversione ed allora viene restituito 0.

es:
se in input l'utente scrive: "punti: 14 9.1 7 21"
mediante la sequenza di istruzioni

```
int i,j,k,result;
double g;
result=scanf("punti: %d  %lf  %d  %d",  &i , &g  , &j , &k)
if(result==EOF)
        printf("FINE FILE INPUT\n");
else
        if(result<4)
                printf("ERRORE, solo %d dati in input\n",result);
        else
                printf("OK: i=%d  g=%lf  j=%d  k=%d\n",i,g,j,k);
```

ottiene a video:
OK: i==14 g==9.1 j==7 k==21

NB
la stringa di formato viene scandita dalla scanf, e se nel formato c'è un blank (uno spazio bianco) l'input può contenere più caratteri di spaziatura blank tab newline ecc. che non vengono considerati (cioè vengono considerati come un unico carattere blank).

ACCESSO AI FILES

Problema: Leggere i dati da un file, e scrivere dei dati su un altro file.

Supponiamo di avere un file c:\users\input.txt di testo formato da linee costituite da due double x e y (ovvio in forma di caratteri ascii).

```
139.2       29.1
-13.1       1009.0
 .......
```

Vogliamo leggere tutte le coppie e scrivere su un file di testo c:\users\output.txt le sole coppie in cui x>0

```
int main(void)
FILE *finput, *foutput;
double x,y;
int result;

finput=fopen("c:\\users\\input.txt","rt");
if ( finput==NULL )
  {  printf("errore: impossibile aprire file di input\n");  exit(0);  }
foutput=fopen("c:\\users\\output.txt","wt");
if ( foutput==NULL )
  {  printf("errore: impossibile aprire file di output\n");  exit(0);  }

/* fino a che non sono alla fine del file di input */
while(  (result=fscanf(finput,"%lf %lf\n", &x, &y)) != EOF )
  {
  if(result != 2) /* ho letto meno campi di quelli richiesti */
    {   printf("errore in lettura\n");   exit(0);   }
  if( x > 0.0 )
    fprintf(foutput,"%f %f\n",x,y);
  }
fclose(finput);
fclose(foutput);
return(1);
}
```

NB. **prima di essere letto o scritto un file deve essere aperto mediante la funzione fopen**

FILE *fopen(char *nomefile, char *modo);

il primo parametro specifica il nome del file ed eventualmente il percorso per raggiungerlo come ad es: c:\\users\\prove\\pippo.txt .\\pippo.txt pippo.txt

ERRORE COMUNE: NOTARE la necessità d'usare la doppia BACLSLASH \\ al posto di una sola

il secondo parametro specifica il modo di apertura del file, che sara' diverso a seconda dell'uso che si vuole fare del file. Il modo puo' essere uno dei seguenti

"**r**" apertura in sola lettura, per leggere, posiziona all'inizio del file.

"**w**" apertura in sola scrittura, crea il file se non esiste, lo cancella e si posiziona all'inizio se gia' esiste.

"**a**" apertura in scrittura per aggiungere dati alla fine del file, se il file esiste gia' si posiziona alla fine del file

per poter aggiungere dati senza cancellare quelli presenti

A questi modi si puo' aggiungere un "+" ottenendo "r+" "w+" "a+" per permettere l'aggiornamento in lettura e scrittura. La posizione in cui ci si colloca e' quella dovuta a "r" o "w" o "a"

A questi modi si puo" aggiungere un "t" per indicare che si apre il file come un file di testo, ed allora saranno utilizzabili le primitive di lettura e scrittura di testo fgets, fputs, oppure un "b" ad indicare che il file verra' aperto come un file binario. Per default il modo di apertura é di tipo testo "t".

La funzione fopen restituisce un PUNTATORE a FILE detto **file pointer** che punta ad una struttura chiamata FILE che 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 questo valore restituito ed utilizzarlo in tutte le altre letture o scritture su file.

Quando il file e' stato aperto, possiamo leggerlo e/o scriverlo a seconda del modo di apertura utilizzando le seguenti funzioni di libreria.

```
int getc(FILE *f);
int putc(int c, FILE *f);

int fscanf(FILE *finput, char *format, ... );
int fprintf(FILE *foutput, char *format, ... );
char *fgets(char *dest,  int nchar, FILE *finput);
```

Quando il file non serve più, deve essere chiuso con la chiamata alla funzione di libreria

int fclose(FILE *f);

che restituisce 0 in caso vada tutto bene, 1 in caso di errore.

INPUT / OUTPUT CARATTERE per CARATTERE da file

int getc(FILE *f); legge il prossimo carattere dal file f.
restituisce il carattere oppure EOF in caso di errore o di fine file.

int putc(int c, FILE *f); scrive il carattere c sulla posizione corrente del file.
restituisce il carattere scritto oppure EOF in caso di errore

INPUT / OUTPUT FORMATTATO da file

int fscanf(FILE *finput, char ***format, ...);** e' analoga alla scanf ma prende input dal file finput invece che dallo standard input
restituisce il numero di campi letti oppure EOF se la posizione corrente nel file e' a fine file

int fprintf(FILE *foutput, char *format, **...**); e' analoga alla printf ma scrive sul file invece che sullo standard output
restituisce il numero di byre scritti

INPUT / OUTPUT di linee da file

la lettura scrittura da un file puo' essere effettuata anche linea per linea usando le seguenti funzioni:

char *fgets(char *dest, int nchar, FILE *finput);

legge dal file finput una linea di input (**compreso il NEWLINE \n**) fino ad un massimo di **nchar-1 caratteri** e la scrive nel vettore dest. Aggiunge dopo l'ultimo carattere un '\0' per terminare la stringa.
restituisce un puntatore alla stringa in cui ha scritto (dest) oppure NULL in caso di errore

int *fput(char *string, FILE *foutput);

scrive sul file foutput una stringa (**con o senza il NEWLINE \n**)
restituisce 0 se va tutto bene, EOF in caso di errore.