usr_29.txt   Per Vim version 7.0. Ultima modifica: 2006 Apr 24

VIM USER MANUAL - di Bram Moolenaar
Traduzione di questo capitolo: Antonio Colombo

Spostarsi attraverso i programmi

Chi ha creato Vim è un programmatore di computer. Non sorprende quindi che Vim contenga molte funzionalità utili nella scrittura di programmi. Spostatevi per trovare dove vengono definiti o usati degli identificatori. Visualizzate dichiarazioni in una finestra apposita. Ulteriori possibilità sono descritte nel prossimo capitolo.

  Capitolo seguente: |usr_30.txt| Editare programmi
Capitolo precedente: |usr_28.txt| La piegatura
             Indice: |usr_toc.txt|

==

29.1 Usare i tag

Cos'è un tag? È un posto dove è definito un identificatore. Un esempio è la definizione di una funzione in un programma C o C++. Una lista di tag è il contenuto di un file di tag. Il file può essere usato da Vim per saltare direttamente da un posto qualsiasi al tag, il posto dove un identificatore è definito.
   Per generare il file di tag per tutti i file C nella directory corrente, usate il comando seguente: **>**

```
ctags *.c
```

[Il nome del file generato è sempre "tags" - NdT]
"ctags" è un programma a parte. Quasi tutti i sistemi Unix lo hanno già installato. Se ancora non l'avete, potete trovare "Exuberant ctags" qui:

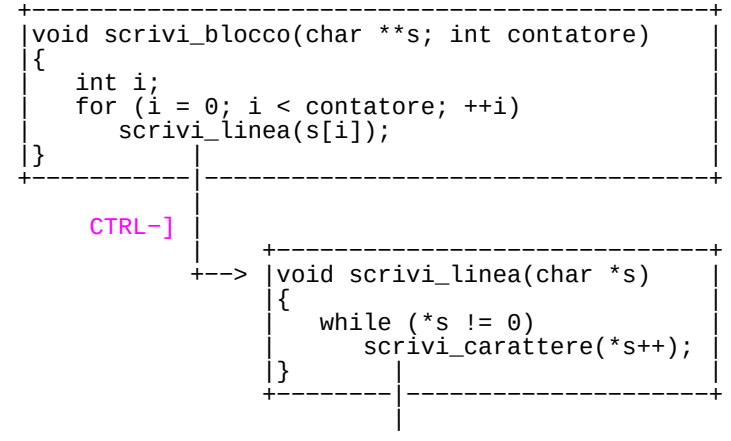
```
http://ctags.sf.net ~
```

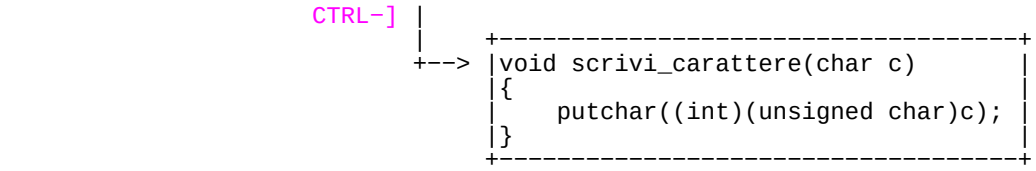
Adesso, quando siete in Vim e volete andare a una definizione di funzione, potete raggiungerla velocemente usando il comando seguente: **>**

```
:tag startlist
```

Questo comando troverà una funzione "startlist" anche se è in un altro file.
   Il comando CTRL-] salta al tag della parola che è sotto il cursore. Questo facilita l'esplorazione di una "giungla" di codice C. Supponete, ad es., di trovarvi in una funzione "scrivi_blocco". Potete vedere che chiama "scrivi_linea". Ma cosa fa "scrivi_linea"? Posizionando il cursore sulla chiamata a "scrivi_linea" e battendo CTRL-], saltate alla definizione di questa funzione.

La funzione "scrivi_linea" chiama "scrivi_carattere". Per capire cosa faccia "scrivi_carattere", posizionate il cursore sulla chiamata a "scrivi_carattere" e battete CTRL-]. Siete ora alla definizione di "scrivi_carattere".





Il comando ":tags" visualizza la lista di tag attraverso cui siete passati:

```
:tags
  # A  tag          DA__  linea in file/testo ~
  1  1 scrivi_linea        8  scrivi_blocco.c ~
  2  1 scrivi_carattere    7  scrivi_linea.c ~
> ~
```

Ora tornate indietro. Il comando CTRL-T torna al tag precedente. Nell'esempio sopra tornate a una funzione "scrivi_linea", nella chiamata a "scrivi_carattere".

Questo comando accetta come argomento un contatore, che indica di quante tag tornare indietro. Siete andati avanti, e poi indietro. Andiamo avanti ancora. Il comando seguente va al primo tag di una lista: >

```
:tag
```

Potete premettergli un contatore e saltare in avanti di alcuni tag. Ad es.: ":3tag". Anche con CTRL-T si può specificare un contatore.

Questi comandi quindi vi permettono di scendere lungo una cascata di chiamate con CTRL-] e di tornare indietro ancora con CTRL-T. Usate ":tags" per vedere dove vi trovate.

DIVIDERE LA FINESTRA

Il comando ":tag" rimpiazza il file nella finestra corrente con quello che contiene la nuova funzione. Supponete di voler vedere non solo la vecchia funzione ma anche quella nuova. Potete dividere in due la finestra usando il comando ":split" seguito dal comando ":tag". Vim ha un comando "scorciatoia" che fa entrambe le cose: >

```
:stag nome_tag
```

Per dividere in due la finestra corrente e saltare al tag sotto il cursore usate questo comando: >

```
CTRL-W ]
```

Se un contatore è specificato, la nuova finestra sarà alta quel numero di linee.

PIU' FILE DI TAG

Quando avete file in molte directory, potete create un file di tag in ognuna di esse. Vim sarà capace solo di saltare a tag all'interno di quella directory.

Per trovare più file di tag, impostate l'opzione 'tags' per includere tutti i file di tag che vi interessano. Esempio: >

```
:set tags=./tags,./../tags,./*/tags
```

Questo trova un file di tag nella stessa directory del file corrente, nel livello di directory superiore e in tutte le sub-directory.

In questo modo di usano parecchi file di tag, ma potrebbero servirene altri. Ad es., modificando un file in "~/proj/src", non trovereste il file di tag "~/proj/sub/tags". In vista di questa situazione Vim permette di cercare un intero albero di directory, per utilizzarne i file di tag. Esempio: >

```
:set tags=~/proj/**/tags
```

UN SOLO FILE DI TAG

Quando Vim deve cercare in parecchi posti per trovare file di tag, potete udire il vostro hard disk che rumoreggia. La faccenda può diventare lunga. In questo caso è meglio impiegare lo stesso tempo per generare un unico grosso file di tag. Potreste lasciar girare un simile lavoro durante la notte.

È necessario il programma "Exuberant ctags", prima citato. Provvede un argomento per cercare attraverso un intero albero di directory: >

```
cd ~/proj
ctags -R .
```

Il bello della faccenda è che "Exuberant ctags" riconosce vari tipi di file. E quindi è possibile far questo non solo con programmi C e C++, ma anche per script Eiffel e perfino per script Vim. Si veda la documentazione di ctags per come funziona la cosa.
Ora dovete solo dire a Vim dov'è il vostro grosso file di tag: >

```
:set tags=~/proj/tags
```

CORRISPONDENZE MULTIPLE

Quando una funzione è definita più volte (o un metodo in parecchie classi), il comando ":tag" salterà alla prima delle definizioni. Se c'è una corrispondenza nel file corrente, quella è usata per prima.
Potete poi saltare ad altre corrispondenze per lo stesso tag con: >

```
:tnext
```

Ripetete questo comando per trovare ulteriori corrispondenze. Se sono tante, potete scegliere quella a cui saltare: >

```
:tselect nome_tag
```

Vim vi farà scegliere da una lista:

```
  # pri tipo tag              file ~
  1 F   f    mch_init         os_amiga.c ~
               mch_init() ~
  2 F   f    mch_init         os_mac.c ~
               mch_init() ~
  3 F   f    mch_init         os_msdos.c ~
               mch_init(void) ~
  4 F   f    mch_init         os_riscos.c ~
               mch_init() ~
Batti n. di scelta (<INVIO> per lasciar perdere): ~
```

Potete ora scegliere il numero (nella prima colonna) della corrispondenza a cui volete saltare. L'informazione nelle altre colonne vi dà una buona idea di dove dove la corrispondenza è definita.

Per muoversi fra i tag corrispondenti, si possono usare questi comandi:

```
:tfirst                 vai alla prima corrispondenza
:[count]tprevious       vai alla [contatore] corrispondenza precedente
:[count]tnext           vai alla [contatore] prossima corrispondenza
:tlast                  vai alla ultima corrispondenza
```

Se [count] è omesso ne viene usato uno solo.

TROVARE NOMI TAG

Usare il completamento della linea comandi permette di evitare di immettere un lungo nome di tag. Immettete solo la parte iniziale e battete <Tab>: >

```
:tag scrivi_<Tab>
```

Vi verrà segnalata la prima corrispondenza. Se non è quella giusta, battete <Tab> finché non trovate quella buona.
Talora conoscete solo parte del nome di una funzione. Oppure avete tante tag che iniziano con la stessa stringa, ma finiscono in modo diverso. Allora potete dire a Vim di usare un'espressione per trovare il tag.
Supponete di voler saltare a un tag che contiene "blocco". Prima battete questo: >

```
:tag /blocco
```

Adesso usate il completamento della linea comandi: battete <Tab>. Vim troverà tutti i tag che contengono "blocco" e userà la prima corrispondenza.
La "/" prima del nome del tag dice a Vim che quel che segue non è un nome di tag, ma un'espressione. Potete usate tutto quel che mettete in un'espressione di ricerca qui. Ad es., supponete di voler selezionare un tag

che comincia con "scrivi_": >

```
:tselect /^scrivi_
```

L'accento circonflesso "^" specifica che il nome del tag inizia con "scrivi_". Altrimenti una corrispondenza verrebbe trovata anche nel mezzo del nome di tag. Allo stesso modo il carattere "$" posto alla fine richiede solo corrispondenze che si trovino nella parte finale del nome di un tag.

UN NAVIGATORE DI TAG

Dato che CTRL-] vi porta alla definizione dell'identificatore sotto il cursore, potete usate una lista di nomi di identificatore come un indice. Ecco un esempio.
Prima create una lista di identificatori (ciò richiede "Exuberant ctags"): >

```
ctags --c-types=f -f funzioni *.c
```

Adesso avviate Vim senza un file, ed editate questo file in Vim, in una finestra separata verticalmente: >

```
vim
:vsplit funzioni
```

La finestra contiene una lista di tutte le funzioni. Ci sono anche altre informazioni, ma potete ignorarle. Battete ":setlocal ts=99" per fare un po' di "pulizia".
In questa finestra, definite la mappatura: >

```
:noremap <buffer> <CR> 0ye<C-W>w:tag <C-R>"<CR>
```

Muovete il cursore sulla linea che contiene una funzione a cui volete saltare. Poi premete <Invio>. Vim passerà all'altra finestra e salterà alla funzione che avete selezionato.

ARGOMENTI CORRELATI

Potete impostare 'ignorecase' per non dover battere maiuscole e minuscole nei nomi di tag.

L'opzione 'tagbsearch' dice se il file di tag è già in ordine alfabetico o no. Il default è di supporre che il file di tag sia già ordinato, cosa che rende la ricerca molto più veloce, ma non funziona se il file di tag non è ordinato.

L'opzione 'taglength' si può usare per dire a Vim il numero di caratteri significativi in un nome di tag.

Quando usate il programma SNiFF+, potete usate l'interfaccia fra SNiFF e Vim. Si veda |sniff|. SNiFF+ è un programma a pagamento.

Cscope è un programma free. Non solo trova i posti dove un identificatore è stato dichiarato, ma anche dove viene usato. Si veda |cscope|.

==
29.2 La finestra di anteprima

Quando modificate del codice che contiene una chiamata a funzione, dovete passargli gli argomenti corretti. Per sapere che valori passare potete guardare a come la funzione è definita. Il meccanismo dei tag si presta molto bene ad effettuare questo controllo. Preferibilmente la definizione viene visualizzata in un'altra finestra. Per questo si può usare la finestra di anteprima.
Per aprire una finestra di anteprima per visualizzare la funzione "scrivi_carattere": >

```
:ptag scrivi_carattere
```

Vim aprirà una finestra, e salterà al tag "scrivi_carattere". Poi vi riporterà indietro alla posizione di partenza. Così potete continuare a modificare il file senza dover ricorrere al comando CTRL-W.
Se il nome di una funzione ricorre nel testo, potete vedere la sua definizione nella finestra di anteprima battendo: >

```
CTRL-W }
```

Esiste uno script che automaticamente visualizza il testo dove è stata

definita la parola sotto il cursore. Si veda |CursorHold-example|.

Per chiudere la finestra di anteprima, usate questo comando: >

```
:pclose
```

Per modificare un file nella finestra di anteprima, usate ":pedit". Questo può tornare utile per modificare un file "header" [contenente definizioni che di solito includono più programmi - Ndt], ad es.: >

```
:pedit definizioni.h
```

Infine, ":psearch" si può usare per trovare una parola nel file corrente e nei file da questo inclusi e visualizzare la corrispondenza nella finestra di anteprima. Questo è particolarmente utile se si usano funzioni di libreria, per le quali non avete un file di tag. Ad es.: >

```
:psearch popen
```

Ciò mostrerà il file "stdio.h" nella finestra di anteprima, col prototipo di funzione per popen():

```
FILE    *popen __P((const char *, const char *)); ~
```

Potete specificare l'altezza della finestra di anteprima, quando è aperta, con l'opzione 'previewheight'.

==
29.3 Muoversi all'interno di un programma

Poiché un programma ha una struttura, Vim può riconoscere parti di esso. Ci sono comandi che si possono usare per muoversi all'interno.
   I programmi C spesso contengono costrutti del tipo:

```
#ifdef USE_POPEN ~
    fd = popen("ls", "r") ~
#else ~
    fd = fopen("tmp", "w") ~
#endif ~
```

Ma molto più lunghi, e possibilmente rientrati. Posizionate il cursore su "#ifdef" e battete %. Vim salterà ad "#else". Battendo % ancora arrivate a "#endif". Un altro % vi riporta indietro a "#ifdef".
   Quando il costrutto è annidato, Vim troverà l'elemento corrispondente. Questa è una buona maniera per controllare se avete dimenticato un "#endif".
   Quando siete nel bel mezzo di un "#if" - "#endif", potete saltare all'inizio dello stesso con: >

```
[#
```

Se non siete dopo un costrutto "#if" o "#ifdef" Vim manda un segnale acustico di errore. Per raggiungere il prossimo "#else" o "#endif" usate: >

```
]#
```

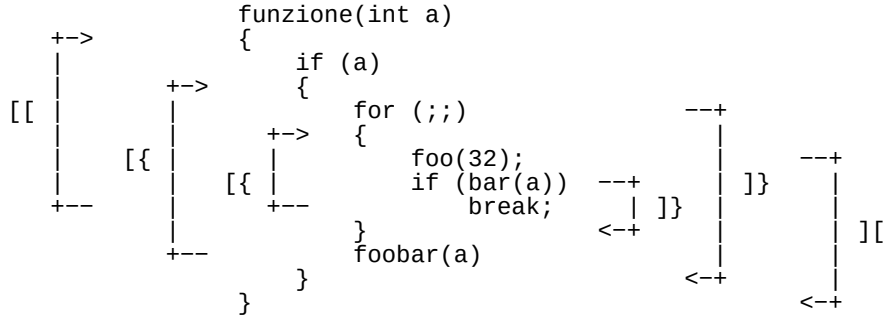
Questi due comandi saltano qualsiasi blocco "#if" - "#endif" che incontrano. Ad es.:

```
#if defined(HAS_INC_H) ~
    a = a + inc(); ~
# ifdef USE_THEME ~
    a += 3; ~
# endif ~
    set_width(a); ~
```

Col cursore sull'ultima linea, "[#" vi porta alla prima linea. Il blocco "#ifdef" - "#endif" nel mezzo viene saltato.

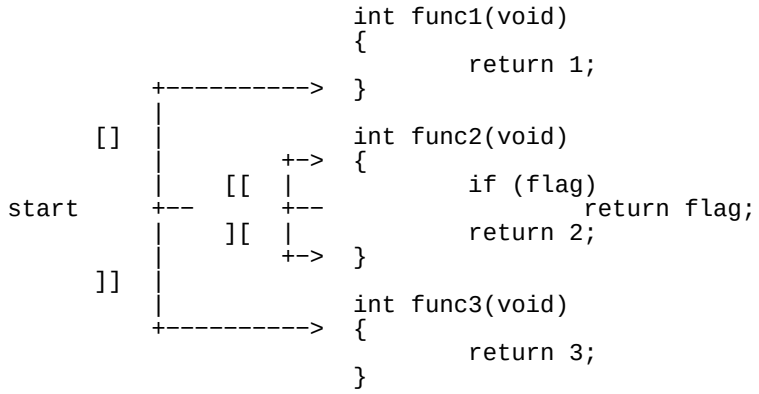
MUOVERSI ALL'INTERNO DI BLOCCHI DI PROGRAMMA

Nel codice C, i blocchi sono racchiusi fra {}. Questi blocchi possono essere anche molto lunghi. Per spostarvi all'inizio del blocco più esterno, usate il comando "[[". Usate "][" per portarvi alla fine. Questi comandi suppongono che "{" e "}" siano nella prima colonna.
   Il comando "[{" vi porta all'inizio del blocco corrente. Coppie di {} allo stesso livello vengono saltate. "]}" salta alla fine.
   Una panoramica:



Quando scrivete in C++ o Java, il blocco più esterno {} è per la "class". Il livello successivo di {} è per un "method". Se siete da qualche parte all'interno di una "class", usate "[m" per trovare il precedente inizio di un "method". "]m" trova la prossima fine di un "method".

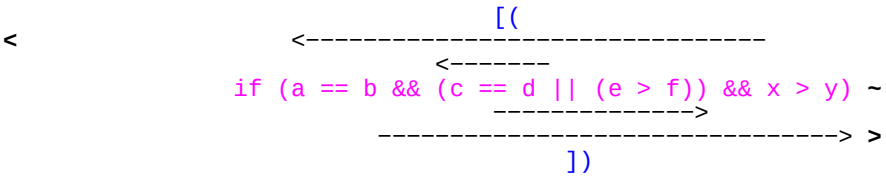
Inoltre, "[]" va all'indietro alla fine di una funzione e "]]" va avanti all' inizio di una nuova funzione. La fine di una funzione è definito da una "}" nella prima colonna.



Non dimenticate che potete anche usare "%" per muovervi tra (), {} e [] corrispondenti. Questo funziona anche quando sono separati da parecchie linee di codice.

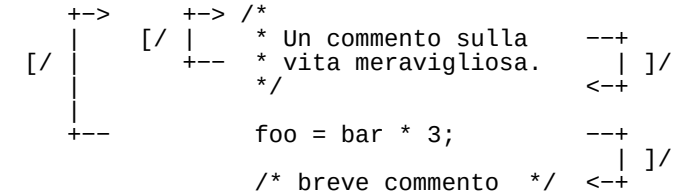
MUOVERSI TRA LE PARENTESI

I comandi "[(" e "])" funzionano in modo simile a "[{" e "]}", solo che si muovono fra coppie di () invece che di {}.



MUOVERSI TRA I COMMENTI

Per muoversi all'indietro verso l'inizio di un commento, usate "[/". Muovetevi in avanti verso la fine di un commento con "]/". Questo funziona solo per commenti delimitati da /* - */.



==========

29.4 Trovare identificatori globali

State modificando un programma C e vi domandate se una variabile è dichiarata

come "int" o "unsigned". Un modo veloce per accertarlo è quello di usare il comando "[I".
Supponete il cursore sia sulla parola "column". Battete: >

```
[I
```

Vim elencherà le linee corrispondenti che riesce a trovare. Non solo nel file corrente, ma anche in tutti i file inclusi (e nei file da questi ultimi a loro volta inclusi, etc.). Il risultato è del tipo:

```
structs.h ~
 1:   29     unsigned     column;    /* column number */ ~
```

Il vantaggio rispetto ad usare i tag o la finestra di anteprima è che i file inclusi sono a loro volta controllati. Il più delle volte si arriva così alla dichiarazione che interessa. Anche quando il file di tag non è aggiornato. Anche quando non esiste un file di tag per i file inclusi.
Perché il comando "[I" funzioni, servono tuttavia alcuni prerequisiti. Innanzitutto, l'opzione 'include' deve specificare come un file è incluso. Il valore predefinito funzione per C e C++. Va cambiato in maniera opportuna, se state modificando file scritti in altri linguaggi.

LOCALIZZARE I FILE INCLUSI

Vim troverà i file inclusi nelle directory specificate con l'opzione 'path'. Se una directory non è listata, alcuni file inclusi non potranno essere trovati. Potete accertarvene con questo comando: >

```
:checkpath
```

Verranno elencati i file inclusi che non si riescono a trovare. Anche i file inclusi nei file che [invece] si riescono a trovare. Un esempio di output:

```
--- File inclusi non trovati nel percorso --- ~
<io.h> ~
vim.h --> ~
  <functions.h> ~
  <clib/exec_protos.h> ~
```

Il file "io.h" è incluso dal file corrente e non può venire trovato. "vim.h" invece può essere trovato, e quindi ":checkpath" lo legge e controlla cosa a sua volta includa. I file "functions.h" e "clib/exec_protos.h", inclusi da "vim.h" non vengono trovati.

```
Note:
Vim non è un compilatore. Non interpreta istruzioni "#ifdef".
Questo significa che ogni "#include" viene controllata, anche
se viene dopo una istruzione "#if NEVER". [Ossia, si
controllano tutte le "#include", sia quelle in uso che quelle
inutilizzate. - NdT]
```

Per correggere i file che non possono essere trovati, aggiungete una directory all'opzione 'path'. Un buon posto per scoprire dove sia il Makefile. Cercate linee che contengano parametri "-I", tipo "-I/usr/local/X11". Per aggiungere questa directory usate: >

```
:set path+=/usr/local/X11
```

Quando ci sono tante subdirectory, potete usare la "wildcard" "*". Ad es.: >

```
:set path+=/usr/*/include
```

Questo potrebbe trovare dei file in "/usr/local/include" e anche in "/usr/X11/include".

Se lavorate su un progetto con un intero albero nidificato di file inclusi, la notazione "**" è utile. Utilizzandola, verranno cercate tutte le sottodirectory. Ad es.: >

```
:set path+=/projects/invent/**/include
```

Così si troveranno file nelle directory:

```
/projects/invent/include ~
/projects/invent/main/include ~
/projects/invent/main/os/include ~
etc.
```

Ci sono anche ulteriori possibilità. Si veda l'opzione 'path' per maggiori informazioni.
   Se voler vedere quali file inclusi sono stati trovati, usate questo comando: **>**

```
:checkpath!
```

Otterrete una (...lunga) lista dei file inclusi, i file che questi a loro volta includono, e così via. Per non allungare troppo la lista, Vim indica "(Già elencato)" per file che sono già stati censiti, e non lista di nuovo tutti i file che questi includono.

SALTARE A UNA CORRISPONDENZA

"[I" produce una lista con solo una linea di testo. Quando volete dare un'occhiata più da vicino al primo elemento, potete saltare a quella linea col comando: **>**

```
[<Tab>
```

Potete anche usare "[CTRL-I", poiché CTRL-I ha lo stesso effetto che battere <Tab>.

La lista che "[I" produce ha un numero all'inizio di ogni linea. Quando volete saltare a un altro elemento diverso dal primo, digitate quel numero prima del comando: **>**

```
3[<Tab>
```

Salterà al terzo elemento nella lista. Ricordate che potete usate CTRL-O per tornare indietro dove vi trovavate.

COMANDI CORRELATI

```
[i              lista solo la prima corrispondenza
]I              lista solo elementi sotto il cursore
]i              lista solo il primo elemento sotto il cursore
```

TROVARE IDENTIFICATORI PARTICOLARI

Il comando "[I" trova qualsiasi identificatore. Per trovare solo macro, definite con "#define" usate: **>**

```
[D
```

Di nuovo, anche i file inclusi vengono controllati. L'opzione 'define' specifica com'è formata una linea che definisce elementi da ricercare tramite "[D". Potreste cambiare questa definizione per farla funzionare con linguaggi diversi da C o C++.
   I comandi correlati a "[D" sono:

```
[d              lista solo la prima corrispondenza
]D              lista solo elementi sotto il cursore
]d              lista solo il primo elemento sotto il cursore
```

==
29.5 Trovare identificatori locali

Il comando "[I" ricerca nei file inclusi. Per cercare solo nel file corrente, e saltare alla prima istruzione dove la parola sotto il cursore è usata: **>**

```
gD
```

Pensate a "gD" come: Go_to Definition [Vai alla definizione]. Questo comando è molto utile per trovare una variabile o funzione che sia stata dichiarata localmente ("static", nel gergo C). Ad es. (il cursore è su "contatore"):

```
   +->   static int contatore = 0;
   |
   |     int get_contatore(void)
gD |     {
   |         ++contatore;
   +--       return contatore;
         }
```

Per delimitare ulteriormente la ricerca, e considerare solo la funzione corrente, usate questo comando: **>**

```
        gd
```

Questo comando partirà dall'inizio della funzione corrente a trovare la prima occorrenza della parola sotto il cursore. In realtà, si posiziona all'indietro fino a trovare una linea vuota sopra la "{" a colonna 1. Da lì viene effettuata una ricerca in avanti dell'identificatore. Ad es. (cursore su "indice"):

```
                int find_entry(char *name)
                {
           +->      int indice;
           |
        gd |        for (indice = 0; indice < lunghezza_tabella; ++indice)
           |            if (strcmp(table[indice].name, name) == 0)
           +--              return indice;
                }
```

==========

Capitolo seguente: **|**usr_30.txt**|** Editare programmi

  vim:tw=78:ts=8:ft=help:norl:

Segnalare refusi a Bartolomeo Ravera - E-mail: barrav at libero.it
          oppure ad Antonio Colombo - E-mail: azc100 at gmail.com